T.C.
SİİRT ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

İHA İMGELERİNDEN BİLGİSAYAR GÖRÜSÜ KULLANILARAK AĞAÇ SAYISI KESTİRİMİ

YÜKSEK LİSANS TEZİ

Ayhan TALAY
(153111011)

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Tez Danışmanı: Doç. Dr. Musa ATAŞ

Haziran-2019
SİİRT

TEZ KABUL VE ONAYI

Ayhan TALAY tarafından hazırlanan "**İHA İmgelerinden Bilgisayar Görüsü Kullanılarak Ağaç Sayısı Kestirimi**" adlı tez çalışması 21/06/2019 tarihinde aşağıdaki jüri tarafından oybirliğiyle Siirt Üniversitesi Fen Bilimleri Enstitüsü Elektrik-Elektronik Mühendisliği Anabilim Dalı'nda YÜKSEK LİSANS TEZİ olarak kabul edilmiştir.

Jüri Üyeleri	**İmza**
Başkan Doç. Dr. İbrahim Berkan AYDİLEK	.....
Danışman Doç. Dr. Musa ATAŞ	.....
Üye Dr. Öğr. Üyesi Melih KUNCAN	.....

Yukarıdaki sonucu onaylarım.

Doç. Dr. Fevzi HANSU
Fen Bilimleri Enstitüsü Müdürü

Bu tez çalışması Siirt Üniversitesi Bilimsel Araştırma Projeleri (SİÜ-BAP) tarafından 2017-SİUFEB-21 nolu proje ile desteklenmiştir.

ÖN SÖZ

Bu tez çalışmasının hazırlanmasında, yürütülmesinde ve oluşumunda ilgi ve desteğini esirgemeyen, engin bilgi ve tecrübelerinden yararlandığım, yönlendirme ve bilgilendirmeleriyle çalışmamı bilimsel temeller ışığında şekillendiren hocam Sayın Doç. Dr. Musa ATAŞ'a sonsuz teşekkürlerimi sunarım. 2017-SİUFEB-21 No'lu proje ile tezimi maddi olarak destekleyen Siirt Üniversitesi Bilimsel Araştırma Projeleri Yönetim Birimi Başkanlığı'na teşekkür ederim. Sevgili aileme maddi manevi hiçbir yardımı esirgemeden yanımda oldukları için tüm kalbimle teşekkür ederim.

Ayhan TALAY
SİİRT-2019

İÇİNDEKİLER

Sayfa

TABLOLAR LİSTESİ

Sayfa

ŞEKİLLER LİSTESİ

Sayfa

ÖZET

YÜKSEK LİSANS TEZİ

İHA İMGELERİNDEN BİLGİSAYAR GÖRÜSÜ KULLANILARAK AĞAÇ SAYISI KESTİRİMİ

Ayhan TALAY

Siirt Üniversitesi Fen Bilimleri Enstitüsü
Elektrik-Elektronik Mühendisliği Anabilim Dalı

Danışman : Doç. Dr. Musa ATAŞ

2019, 50 Sayfa

Günümüzde İnsansız Hava Araçları (İHA), birçok alanda (askeri, tarım, güvenlik, izleme, acil yardım, turizm) kullanılmaktadır. Bu tez çalışmasında, İHA'dan elde edilen yüksek çözünürlüklü renkli KYM (Kırmızı, Yeşil, Mavi) görüntülerden Siirt Üniversitesi Kezer Kampüsü alanındaki bazı bölgelerin ağaçların sayısının kestirimi gerçekleştirilmiştir. Kampüs alanı içerisinde belirlenen bölgelerde İHA ile havadan alınan görüntülerin ortofoto haritası oluşturulduktan sonra bu ortofoto haritalara ait HSV (Hue Saturation Value), RGB (Red Green Blue) ve GRAY (Gri) formatlarında her nesneye ait ilgi bölgesi ROI (Region of Interest) belirlenmiştir. İlgi bölgelerinden öğrenme, doğrulama ve test veri kümelerini içeren bir veri seti oluşturulmuştur. Oluşturulan bu veri seti çeşitli makine öğrenmesi algoritmaları kullanılarak ağaçların tespiti ile ilgili sınıflandırma başarısı test edilmiştir. Son aşamada ise ağaçların gerçek sayıları manuel yöntemle elde edilerek bir referans modeli oluşturulmuş ve otomatik yöntemle elde edilen ağaçların sayıları ile referans veri karşılaştırılarak doğruluk analizleri yapılmıştır. Geliştirilen yazılımın başka nesnelerin; örnek olarak araç, insan, bina sayısının hesaplanmasında ileride yapılacak projelerde kullanılabileceği öngörülmektedir.

Anahtar Kelimeler: Ağaç Sayısı Kestirimi, Bilgisayar Görüsü, Görüntü İşleme, Makine Öğrenmesi, İHA.

ABSTRACT

MS THESIS

ESTIMATION OF NUMBER OF TREES USING COMPUTER VISION FROM UAV IMAGES

Ayhan TALAY

The Graduate School of Natural and Applied Science of Siirt University
The Degree of Master of Science
in Electrical-Electronics Engineering

Supervisor : Assoc. Prof. Dr. Musa ATAŞ

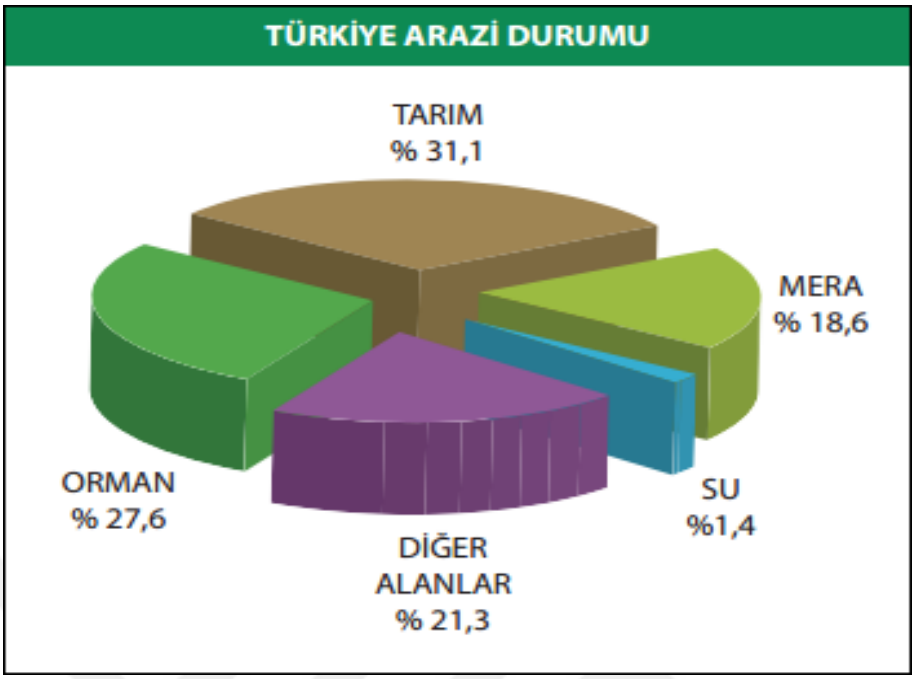
2019, 50 Pages

Unmanned Aerial Vehicles (UAV) are used in many fields such as military, agriculture, security, monitoring, emergency aid and tourism. In this thesis, the number of trees of some regions in Siirt University Kezer Campus area is estimated from high resolution colored RGB (Red, Green, Blue) images obtained from UAV. After the orthophoto map of the images taken from the air with the UAV in the designated areas within the campus area, the region of interest of each object was determined in HSV (Hue Saturation Value), RGB (Red Green Blue) and GRAY (Gray) formats of these orthophoto maps. A data set was created containing learning, validation and test datasets from regions of interest. This data set was tested by using various machine learning algorithms. In the last stage, the real numbers of the trees were obtained by manual method and a reference model was formed and the accuracy analysis was made by comparing the numbers of the trees obtained with the automatic method and the reference data. The developed software can be used in future projects for calculating the number of other objects such as vehicles, people and buildings.

Keywords: Computer Vision, Image Processing, Machine Learning, Tree Number Estimation, UAV.

1. GİRİŞ

Şekil 1'deki verilere göre Türkiye'deki arazilerin yaklaşık %27,6'sı ormanlık alan olarak kabul edilmektedir. Bilindiği gibi, ormanlar gerek ülke ekonomisi gerekse de temiz ve sürdürülebilir bir ekosistem açısından son derece önemlidir. Milli bir servet olan ormanlarımızda ağaç envanterinin oluşturulması ileriye yönelik sürdürülebilir ağaçlandırma politikalarının geliştirilmesinde ve orman varlığını korumada hayatî önem taşımaktadır. Yapılacak ağaç envanter çalışmaları ile bölgenin koşullarına ve ihtiyaçlarına göre mevcut ağaç türlerinin sayısı, bireysel özellikleri, konumu, sağlık ve bakım durumlarına yönelik veriler elde edilmesi mümkündür. Yılmaz ve ark. (2015)'te orman alanlarının sınırları, ağaçların türleri, sayıları, yükseklik ve konum bilgileri gibi bilgilerin tespiti şehir plancılığı, 3B kent modelleme, ormancılık ve tarım faaliyetleri gibi birçok uygulamaya olanak sağlamaktadır. Bu amaçla yapılan yersel ölçme yöntemleri oldukça fazla zaman ve iş gücü kaybına neden olmaktadır. Günümüzde ağaç sayısının hesaplanması işlemi, insan gözlemine ve işgücüne dayanan maliyeti ve hata oranı yüksek bir işlemdir. Dahası, ağaçların tek tek görevli personel tarafından sayılması zaman alıcı ve zahmetli olup bazı bölgeler için tehlike arz etmekte, hata yapma riski taşımaktadır. Diğer taraftan bazen de istatistikî yöntemler kullanılarak belirli bir alandaki ağaç yoğunluğu esas alınıp seçilen coğrafî bölgede bir genelleştirmeye gidilebilmektedir. Bu yaklaşım, sağladığı kolaylıklara paralel olarak hata oranının artmasına da neden olabildiğinden söz konusu hesaplamaların temkinli ve dikkatli bir şekilde yapılması gerekmektedir.



Şekil 1. 1. Türkiye arazi durumu yüzdelik dilimleri

Yukarıda ifade edilen problemleri giderebilmek için görüntü işleme ve makine öğrenmesi kullanarak ağaç sayımı işlemini yapmak mümkündür. Bunun için temel gereksinimler, İHA ve İHA'dan elde edilen görüntülerin işlenmesi sonucu ağaç sayısını hesaplayacak yazılım şeklinde özetlenebilir. Gelişen teknoloji ile artık insansız hava araçlarının (İHA) kullanımı yaygınlaşmıştır. Birçok alanda (askeri, acil yardım, tarım, izleme, güvenlik vb.) İHA'lar sivil havacılık genel müdürlüğünden alınan lisans ve yetki çerçevesinde kullanılmaktadır. Bu çalışma, DJI Advanced 4 Pro İHA kullanılarak bölgenin havadan belirli bir yükseklikte taranması ve çekilen görüntülerin yer istasyonunda gerekli yazılımlarla analiz edilmesi esasına dayanmaktadır. Erişim kolaylığı açısından pilot uygulama alanı olarak, Şekil 2'de gösterildiği gibi Siirt Üniversitesi Kezer Yerleşkesi'nde bulunan 4 bölge seçilmiştir. Belirlenen bölgelere ait alanlar bölütlenerek, İHA taraması bu bölgeler üzerinde gerçekleştirilmiştir. Ayrıca geliştirilen yazılımın doğruluğunu hesaplayabilmek için bu bölgelerde çalışma yapılmış olup, bölgelerdeki ağaç sayıları ve envanter listesi manuel olarak kaydedilmiştir.

İHA'dan elde edilen imgeler görüntü hizalama ve ekleme yöntemleri ile birleştirilip taranan alan düzeltilerek ortofoto harita elde edilmiştir. Yüksek çözünürlük ve boyuttaki söz konusu görüntü, maliyet etkin işlenebilmesi adına (bellek açısından)

ızgara tabanlı küçük resim parçalarına bölünmüştür. Geliştirilen çok kanallı (multi-threaded) uygulama ile söz konusu imgeler, çeşitli görüntü işleme teknikleri yardımıyla analiz edilmiştir. Analiz sonucunda dinamik ilgi alanları DROI (Dynamic Region of Interest) manuel olarak belirlenmiş bölgeler, görüntü matrisinden kesilerek (cropping) makine öğrenmesi algoritması için veri setleri oluşturulup bu DROI'deki piksel yoğunluğu değerleri temelinde hazırlanmıştır. Veri kümesi hazırlanırken ağaç olan bölgelerle olmayan bölgelerin sayıları dengeli olacak bir şekilde ayarlanmıştır. Veri kümesi kendi içerisinde öğrenme-doğrulama-test alt kümelerine ayrılmıştır. Çeşitli makine öğrenme algoritmaları (KNN, LDA, MLP, SVM vb.), veri kümesi üzerinde eğitilip çapraz doğrulama ile optimize edildikten sonra test kümesi üzerinden sınıflandırma başarımları ortaya çıkarılmış olup en iyi makine öğrenmesi algoritması bulunmaya çalışılmıştır. Belirlenen algoritma, yeni İHA görüntüleri üzerinde sınıflandırma yaparak belirlenmiş bölgelerdeki ağaç sayısını, bilgisayar görüsü temelinde tahmin etmiştir. Elde edilen sonuçlar, gerçek sonuçlar ile karşılaştırılıp geliştirilen İHA tabanlı ağaç sayım sisteminin başarısı hesaplanmıştır.

1.1. Tezin Amacı

Bu tez çalışmasının amaçları aşağıda maddeler halinde özetlenmiştir.

- T.C. Tarım ve Orman Bakanlığı ve ilgili kurumların ağaç tespiti ve ağaç envanteri oluşturması çalışmalarına yardımcı olabilecek bir sistemin yazılım alt yapısını oluşturmak.
- İHA ile elde edilen yüksek çözünürlüklü renkli RGB (Red-Green-Blue) görüntülerden otomatik ağaç sayısı tespiti için bir yaklaşım geliştirmek.
- Önerilen otomatik ağaç sayısı tespiti yaklaşımını farklı bölgelerdeki ağaçların yer aldığı test alanlarında test etmek ve sonuçları irdelemek.

1.2. Tezin Organizasyonu

Tez çalışması, toplam 5 ana bölümden oluşmaktadır. Tezin 2. bölümünde literatür araştırması, 3. bölümde materyal ve metot, 4. bölümde bulgular ve tartışma, 5.bölümde sonuç ve öneriler getirilmiştir.

2. LİTERATÜR ARAŞTIRMASI

Literatür araştırması bölümünün ilk kısmında İHA ve ilgili teknolojiler açıklanmış olup, sonraki bölümlerde ise literatürdeki çalışmalara atıflarda bulunulmuştur.

2.1. İnsansız Hava Aracı

İnsansız hava araçları (İHA); değişik boyutlarda ve şekillerde tasarlanmış uçuş için herhangi bir pilota ihtiyaç duymaksızın yerden kontrol edilebilen hava araçlarına verilen genel bir isimdir (Şekil 2.1). Bu sistemler, ilk olarak askerî amaç için tasarlanmış olup hafif dijital kameralar taşıyabilmektedir. Bu sayede fotogrametrik amaçlı hava görüntüleri elde edilebilmektedir. Son yıllarda düşük maliyeti nedeniyle sivil amaçlı kullanımı yaygın hâle gelmiştir. Ancak dünyanın farklı ülkelerinde insansız hava araçlarının kullanımına ilişkin yasal mevzuat hâlâ yetersizdir. Bu sistemler, çoğunlukla arkeolojik çalışmalar gibi özel amaçlar için küçük alanlarda kullanılmaktadır.

Şekil 2. 1. Çalışmada kullandığımız DJI Phantom 4 Advanced İnsansız Hava Aracı

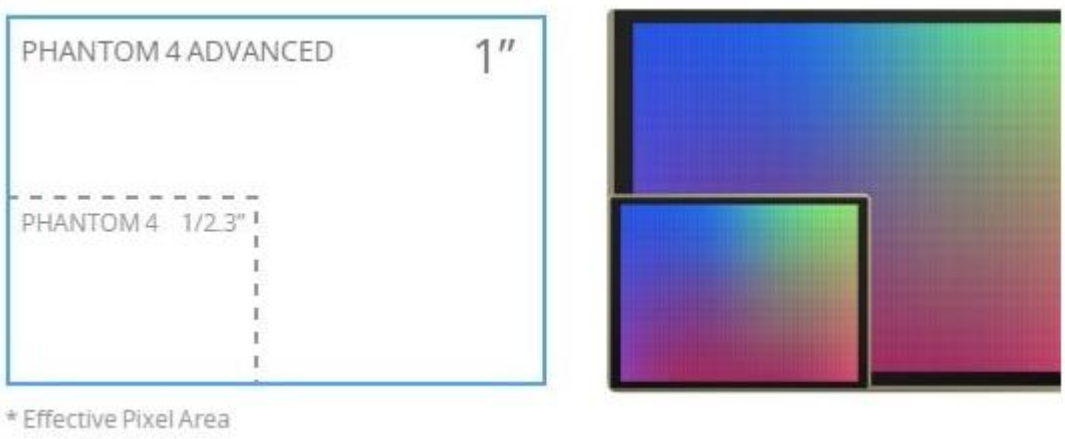
Ülkemizde 500 gramın üzerinde ağırlığı olan tüm İHA'lar için yasal uçuş belgesinin ve lisansının sivil havacılık genel müdürlüğü onaylı olması bir yasal gerekliliktir. 500 gr altındaki İHA'larla gerçekleştirilecek uçuşlarda herhangi bir izin aranmamaktadır. Bu tez çalışması kapsamında tam otomatik uçuş yeteneğine sahip DJI

firmasının ürettiği DJI Phantom 4 Advanced (Şekil 2.1) isimli insansız hava aracı kullanılmıştır. Fotoğraf çekim işlemi, araca gimbal ile sabitlenmiş 20 MP çözünürlüklü (Şekil 2.2) dijital kamera ile gerçekleştirilmiştir.

Şekil 2. 2. İnsansız hava aracına ve kameraya ait detaylı teknik bilgiler aşağıdaki alt bölümlerde işlenmiştir

2.1.1. 1 inch CMOS Sensörlü 20 MP Dijital Kamera

Odak uzaklığı F2.8 geniş açı objektif değerine sahip kamera kaliteli fotoğraf çekebilmek adına 24 mm eş odak uzaklığına ve 8 katmandan meydana gelen kaliteli bir lense sahiptir. Bunların dışında çekilen görüntüleri hareket kaynaklı bozulmalara (motion blur) karşı koruyan dâhili jiroskopa bağlı iki eksenli gimbal ile birlikte mekanik denklanşör sistemi ile donatılmıştır. Bununla birlikte, resmin kalitesine doğrudan etki eden ve foton kazanımını maksimize ederek daha geniş dinamik aralıkta piksel değerlerini kodlayabilen 1 inch büyüklüğündeki CMOS sensör, bir önceki DJI PHANTOM 4 modeline göre daha kaliteli ve yaklaşık 4 kat daha yüksek çözünürlüklü resim çekilmesine olanak tanımaktadır (Şekil 2.3).



Şekil 2. 3. Phantom 4 Advanced ve Phantom 4 çözünürlük karşılaştırması

2.1.2. DJI Phantom 4 Advanced Özellikleri

Tez çalışması süresince kullandığımız DJI Phantom 4 Advanced Pro, yaklaşık 1368 gr ağırlığındadır. Fabrika verilerine göre, 500 gr faydalı yük kaldırma kapasitesine sahip olup maksimum 6000 feet (1829 metre) irtifaya çıkabilmektedir. Ayrıca, 45 m/s hıza ve 5879 mAh bataryası ile 30 dakika uçuş süresine sahiptir.

Bütün bir sistem olarak tasarlanan İHA'lar ana hatlarıyla 4 ana bölümden oluşmaktadır.

1- Operatörler tarafından kullanılan bir kontrol paneli
2- Değişik türlerde üretilen ve yük taşımaya olanak sağlayan hava taşıtı
3- Kontrol paneli ve hava aracı arasında karşılıklı olarak iletişimi sağlayan iletişim sistemi (genellikle bu iletişim için radyo dalgaları kullanılmaktadır).
4- Bakım ve taşıma cihazları gibi kısımların yer aldığı destek ünitesi

Bu araçlar, üzerinde İHA platformu taşıma kapasitesine ve özelliklerine bağlı olarak, video kamera, termal ya da kızılötesi kamera sistemleri, multispektral kameralar, LiDAR algılayıcıları veya bu teknolojilerin birkaçını bir arada sunacak şekilde donatılmış olabilir. İHA'ların havadan alınan görüntülere nazaran bulut ve benzeri olumsuz hava şartlarından etkilenmemesi ve maliyetinin uygun olması gibi bir takım avantajları da bulunmaktadır.

Genel olarak İHA'ları altı grupta sınıflandırabiliriz. İlk grupta uzun menzillerde keşif ve gözetleme yapabilen yaklaşık 15.000 m irtifa ve 24 saat (ve üzeri) seyir süresine sahip İHA'lar bulunmaktadır. İkinci grupta ilk gruba nazaran daha basit sistemler olan 2400 m ile 3000 m arasında irtifaya ve 12 saat seyir süresine sahip

İHA'lar bulunur. Üçüncü grupta çeşitli sivil ve askerî amaçlar için tasarlanan kısa menzilli İHA'lar yer alır. Bu taşıtlar genellikle 100 km'ye kadar menzile sahip olup keşif ve gözetleme gibi görevler için kullanılırlar. Dördüncü grupta 30 km menzili olan mini İHA denilen araçlar yer alır ve bu araçlar elle fırlatılabilme kabiliyetine ve birkaç saatlik seyir süresine sahiptir. Beşinci grupta kentsel ortamlarda, özellikle binaların arasında yavaş uçma ve süzülme gibi özelliklere sahip olan 10 km menzilli yaklaşık 1 saatlik uçuş süreli İHA'lardır. Altıncı ve son grup ise uzunluğu 5 cm' den küçük olan çok kısa menziller için gözetleme yapabilen araçlardır.

2.2. İHA'nın Kullanım Alanları

İnsansız hava araçlarının kullanım amaçları gelişen teknolojilerle birlikte hem sivil hem de askerî amaçlı olarak iki ayrı kategoride incelenebilir. Aşağıda İHA'nın sivil amaçlı uygulamaları maddeler halinde listelenmektedir:

- Havadan haritalama çalışmaları
- Tarımsal uygulamalar
- Sahil koruma
- Çevre koruma
- Gümrük ve Vergilendirme
- Enerji sektörü
- Yangın tespiti ve ormancılık faaliyetleri
- Balıkçılık
- Madencilik ve Arkeolojik çalışmalar
- Hava durumu tahmini
- Trafik kontrol
- Acil durumlar, arama-kurtarma, yangınla mücadele, afet operasyon yönetimi
- Boru hatlarının, su kaynaklarının izlenmesi, su seviyesi değişimlerinin tespiti, su kirliliği tespiti vb.

Askerî amaçlı olarak ise İHA'ların en önemli görevleri istihbarat, gözetleme ve keşiftir. Son zamanlarda SİHA olarak adlandırılan, savaş maksadıyla geliştirilen İHA'lara da rastlamak mümkündür. Bu tür amaçlar için kullanılan hava taşıtlarında görsel ya da ısıl algılayıcılar kullanılmaktadır. Keşif ve gözetleme faaliyetleri için kullanılan hava taşıtlarının yaklaşık 10 km ila 200 km menzilde ve 1 ila 8 saat

arasında görev yapması beklenmektedir. Ayrıca taarruz, hedef benzetimi ve elektronik harp gibi uygulamalarda da İHA'lar askeri amaçlı olarak kullanılabilmektedir.

2.3. İHA Teknolojisindeki Gelişmeler ve Uygulamalar

Teknolojik gelişmelere bağlı olarak, uçuş kontrol ünitelerinde açık kaynak kodlu sistemlerin tercih edilmesi, GPS, jiroskop, manyetometre ve barometre sensörlerini de içeren ataletsel ölçüm birimi (IMU, Inertial Measurement Unit) donanım parçalarının daha ekonomik seviyelere gelmesi nedeniyle İHA üretim maliyetleri hızlı bir düşüş yaşamıştır. Bununla birlikte daha az maliyetle ve hafif multispektral ve hiperspektral kameraların üretilmesi İHA'ların özellikle hassas tarım uygulamaları, gözetleme ve kurtarma gibi önemli alanlarda kullanılmasının önünü açmıştır. Ayrıca GNSS (Global Navigation Satellite System) teknolojisinde kullanılan basit donanım çözümleri sayesinde PPK (Post Processed Kinematic) navigasyonun hesaplama işlemleri daha doğru yapılabilmektedir (Torun, 2017; Colomina ve ark., 2014).

İHA'nın global ölçekte en yaygın olarak uygulama alanı bulduğu ilk beş sektör; emlak ve inşaat, fotogrametri (havada ölçme), film/hava fotoğrafçılığı, tarım ve havadan izleme (takip) alanlarıdır. İHA fotogrametrisi; özellikle tarım, madencilik, havadan fotoğraflama, emlak sektörlerinde kendine yer edinmiştir (Torun, 2017).

2.4. İHA Fotogrametrisi ile Ölçme

İHA fotogrametrisi ve geleneksel fotogrametri arasındaki farklılıklar mevcuttur. İHA fotogrametrisi, lokal çözüm ve lokal optimizasyon bağlamında doğru sonuçlardan çok verimlilik ve tüm veriyi kullanmak üzerine tasarlanmıştır. Öte yandan, geleneksel fotogrametride ise global tutarlılık, model geçerliliği, ölçülerin doğru, tutarlı ve uyumlu olması çerçevesinde işlemler yürütülmektedir (Torun, 2017). İHA fotogrametrisinin klasik fotogrametri yerine geçebilmesi için iki alanda gelişmelerin olması gereklidir. Birincisi, İHA fotogrametrisinde kullanılan matematik/istatistik model ve bunların uygulama noktasının geleneksel fotogrametri ile adaptasyonudur. İkincisi ise klasik fotogrametrinin çalışmasının temelini oluşturan sensör kamera merceği yapısı ve mercek çarpıtması (distortion) bilgisi ile fiziksel koşulların matematik model tasarlamaya sağladığı imkanların, İHA sensörleri için de kullanılabilmesidir (Torun, 2016, 2017; Cryderman ve ark., 2015; Draeyer ve Strecha , 2014).

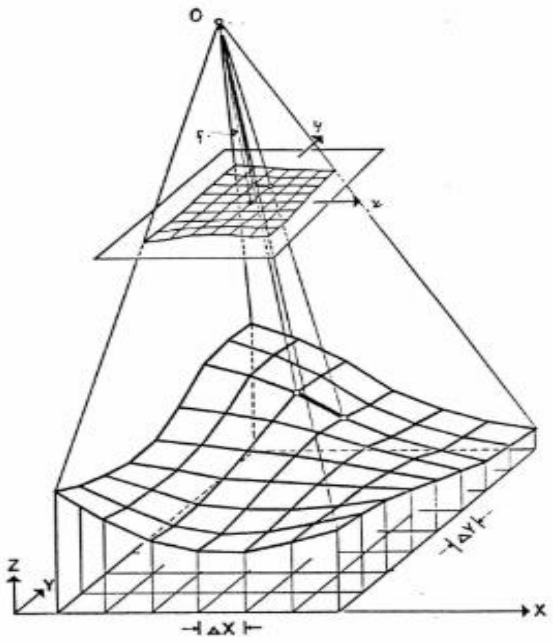
2.5. Ortofoto Nedir

Fotogrametri tekniğiyle fotoğraftaki optik hatalar giderilerek, gerçek düzlemdeki objenin 3 boyutlu koordinatlarının merkezsel iz düşüm yöntemine göre otomatik olarak hesaplanıp, birim alanda yapılan yataylama sonrası oluşan, ölçekli ortoganal fotoğrafa "doğru (true) **ortofoto**" denilmektedir. Resimlerdeki eğiklik etkileri ve yükseklik farklarından (rölfey kayma) ileri gelen hataların giderilmesi ile elde edilen ortofoto görüntüler üzerine; eş yükseklik eğrileri, yükseklik bilgileri, harita kenar bilgileri de eklenmek suretiyle elde edilen ortofoto görüntü parçacıklarının birleştirilmesiyle standart veya rastgele ölçeklerde üretilen foto haritadır. Ortofoto haritaların üretiminde, özel kartografik işlemler olarak fotografik kenar zenginleştirmesi, renk ayrımı ya da bunların bir kombinasyonu gibi birleşik işlemler de uygulanabilmektedir. Bu haritalar ortofolar kamera sistemine sahip uydu ve hava araçları vasıtasıyla elde edilebilmektedir. Ortofoto harita elde etmenin en iyi yollarından biri, hava araçlarını kullanmak olmasına rağmen bu işlemde bazı teknik sorunlar ile karşılaşılmaktadır. Uçak ya da benzeri hava taşıtları ile alınan görüntülerin mümkün olduğunca yeryüzüne göre dikey konumda olması istenir. Ancak uygulamada hava taşıtının eğiklik ve dönüklüğünden dolayı bu konum mümkün olamamaktadır. Bu eğikliğin genellikle 1°' den az olması ideal olup nadiren 3°' yi aştığı durumlar vardır. Dolayısı ile hava fotoğrafları, bu tür istenmeyen eğikliklere sahiptir.

Hava fotoğraflarındaki bir diğer sorun ise 'yükseklik kayması' denilen problemdir. Bu hata, arazideki yapay ya da doğal nesnelerin yükseklik farkından oluşmaktadır. Yükseklik kayması noktaların resim orta noktasından dışa doğru olan kayıklığıdır. Resim orta noktasından uzaklaştıkça yükseklik kayması artar. Bu hataların haricinde hava fotoğraflarında; film ve fotoğraf düzleminden kaynaklanan problemler (filmin büzülmesi, odak uzaklığı hatası vb.), kamera lenslerinde meydana gelen çarpıklıklar, atmosferik yansımadan kaynaklanan çarpıklıklar ve yeryüzü eğikliğinden kaynaklanan çarpıklıklar gibi problemler de oluşabilmektedir. Uzaktan algılanmış görüntüdeki nesneler çeşitli nedenlerden dolayı (sensör eğiklikleri ve yükseklik farkından kaynaklanan kaymalar) gerçekte olması gereken yerlerde olmayabilir (Yılmaz ve ark., 2013; Tao ve ark., 2004). Bu durum, ortorektifikasyon işlemi kullanılarak giderilmektedir. Diğer bir tanımla ortofoto, geometrik olarak rektifiye edilmiş bir hava fotoğrafıdır ya da başka bir tanımla; uçuş yüksekliği, kamera merceği ve fotoğraf alımı sırasındaki kamera eğiklikleri sebebiyle meydana gelmiş

olan hataların düzeltilmiş olduğu hava fotoğraflarıdır. Şekil 2.4.'teki örnekte temsil edildiği üzere ortofoto üretiminde başlıca iki veri seti kullanılmaktadır (Gürbüz, 2016). Bunlar; düzeltilmiş görüntüler (iç ve dış yöneltmesi yapılmış) ve sayısal yükseklik modelidir. Eğer yeryüzünü iyi bir şekilde temsil etmeyen bir sayısal yükseklik modeli kullanılırsa ortofotoda bir takım bozulmalar meydana gelebilir (Gürbüz, 2016).

Bir fotoğrafın tüm bilgilerini içeren ortofotolar, çizgisel haritalarda bulunan topografik elemanlar ile arazi kullanımı ve arazi örtüsü arasındaki ilişkiyi sağlayan detaylara da ulaşmamıza olanak sağlar. Dolayısıyla, yatay mesafe, açı, konum, alan ölçümleri vb. bilgilere imkân sağlayan ortofotolar, harita gibi kullanılabilirler (Gürbüz, 2016).



Şekil 2. 4. Hava fotoğrafı, sayısal yükseklik modeli ve ortofoto (Gürbüz, 2016)

2.6. Önceki Çalışmalar

Moranduzzo ve Melgani (2013) yılında yaptıkları bir çalışmada insansız hava araçları (İHA) ile elde edilen görüntülerde otomobil tespit ve sayma problemini çözmek için bir çözüm sunmuşlardır. İHA görüntüleri çok yüksek bir uzamsal çözünürlükle (birkaç santimetre sırayla) ve sonuç olarak uygun otomatik analiz yöntemlerini gerektiren çok yüksek bir ayrıntı düzeyiyle karakterize edilir. Önerilen yöntem,

otomobilleri tespit edebilecek alanları sınırlandırmak ve böylece yanlış alarmları azaltmak için asfaltlanmış bölgelerin bir tarama adımıyla başlar. Ardından, düşünülen görüntüde bir takım kilit noktaların tanımlandığı skaler değişmez özellik dönüşümüne dayanan bir özellik çıkarma işlemi gerçekleştirir. Ardından, bir destek vektör makinesi sınıflandırıcısı aracılığıyla, otomobillere ve diğer görüntülere atanan kilit noktalar arasında ayrım yapar. Metodun son adımı, "bir kilit nokta-bir otomobil" ilişkisi elde etmek için aynı araca ait kilit noktaların gruplandırılmasına odaklanmıştır. Son olarak, sahada bulunan araç sayısı, belirlenen son kilit nokta sayısına göre verilmiştir. 2 cm uzamsal çözünürlüğü ile karakterize gerçek bir İHA sahasında elde edilen deney sonuçları, önerilen yöntemin ümit verici bir araba sayma doğruluğu gösterdiğini kanıtlamıştır.

Bazi ve ark. (2014) yılında yapılan başka bir çalışmada İHA görüntülerinde palmiye ağaçlarını saymak için otomatik bir yöntem geliştirilmiştir. İlk olarak, Ölçekli Değişmeyen Özellik Dönüştürmesi'ni kullanarak bir dizi kilit nokta ayıklanmıştır. Ardından, bu kilit noktaları, bir dizi palmiye olan ve olmayan önceden eğitilmiş bir Ekstrem Öğrenme Makinesi (ELM) sınıflandırıcısıyla analiz edilmiştir. Çıktı olarak, ELM sınıflandırıcısı algılanan her palmiye ağacını birkaç anahtar noktayla işaretler. Daha sonra, her ağacın şeklini yakalamak için, bu kilit noktaları seviye setlerine dayalı aktif bir kontur yöntemiyle birleştirmeyi önermiştir. Son olarak, palmiye ağaçlarını diğer bitkilerden ayırmak için veri setleri tarafından elde edilen bölgelerin dokusu, yerel ikili desenlerle analiz edilmiştir. Bir palmiye çiftliğinde elde edilen İHA görüntüsünde elde edilen deneysel sonuçlar bildirilmiş ve tartışılmıştır. İHA teknolojisindeki gelişmeler ve veri işleme yetenekleri, orman özelliklerini izlemek ve değerlendirmek için kullanılabilecek yüksek çözünürlüklü görüntü ve üç boyutlu (3D) veri elde etmeyi mümkün kılmıştır.

Mohan ve ark. (2017) yılında yapılan başka bir çalışmada İHA'ya bağlı düşük tüketici sınıfı kameraların uygulanabilirliği ve İHA'dan elde edilen Kanopi Yüksekliği Modellerinde yerel bir maxima tabanlı algoritma kullanarak otomatik bireysel ağaç tespiti için hareket-yapı algoritmasının uygulanabilirliğini değerlendirmişlerdir. Her arsa için referans olarak İHA'dan türetilmiş ortomozaik kullanılarak ağaç sayısı manuel olarak sayılmıştır. Çalışmanın bir parçası olarak toplam 367 referans ağacı sayılmış ve algoritma % 85'ten daha yüksek bir doğrulukla sonuçlanan 312 ağacı tespit etmiştir (F-skoru 0.86). Genel olarak, algoritma 55 ağacı kaçırmış ve hatalı bir şekilde toplamda 358 ağaç sayısıyla sonuçlanan 46 ağaç tespit edilmiştir. Ayrıca sabit ağaç pencere

boyutlarının ve sabit düzleştirme pencere boyutlarının bireysel ağaç tespiti doğruluğu üzerindeki etkisini belirlenmiş ve ağaç yoğunluğu ile sabit ağaç pencere boyutları arasında ters bir ilişki tespit edilmiştir.

Ağaç sayımı, verim tahmini ve izlenmesi, tekrarlama ve yerleşim planlaması vb. için önemli ve gerekli bir uygulamadır. Ancak, bu işlemin arazi koşullarında yapılması pahalı ve yoğun emek gerektiren bir uygulamadır. Ayrıca kişilerden kaynaklanan hatalara sıkça rastlanılmaktadır. Birçok arazinin hesaplanması, toplam alanı hektar başına avuç sayısıyla çarparak rakamları tahmin etme yöntemiyle elde edilmiştir. Bu yöntem, arazi yüzeyinin heterojenliği (tepelik, dalgalı veya düz) ve özellikleri (nehir, arazi veya orman) nedeniyle verimli ve etkin bir yaklaşım değildir. Uzaktan algılama yöntemi, kuş bakışı plantasyona bakış ve ağaçları otomatik olarak sayma, yukarıda ifade edilen soruna bir çözüm olabilir. Birçok plantasyonda dönüm başına düşen ağaç sayısı göz önünde bulundurularak toplam alan tahmin edilmektedir. Ancak arazi yapısının homojen olmaması nedeniyle bu yaklaşım, kesin ve doğru çözümler sunmayabilir. Buna karşılık uzaktan algılama yaklaşımı, kuşbakışı bir görüş sağladığı ve ağaçları otomatik saydığı için bu soruna daha kesin bir çözüm sunmaktadır. Bu yaklaşımda genel anlamda, otomatik ağaç sayma; nesne segmentasyonu, sınıflandırma ve tanımlama gibi görüntü işleme tekniklerini içermektedir. Ancak en iyi algoritma veya teknik hakkında karar vermek zordur. Bu teknikte çok kesin sonuçlar elde edilmesi, girdi olarak kullanılan görüntülerin iyi kalitede ve çok yüksek çözünürlükte olmasına bağlıdır.

Shafri ve ark. (2011) yüksek uzamsal çözünürlüğe sahip (1 m) havadaki hiperspektral verileri kullanarak, otomatik ağaç sayma işlemini gerçekleştirmek için doku analizi, kenar geliştirme, morfoloji analizi ve kabarcık analizi gibi çeşitli yaklaşımları kullanımı amacıyla bir şema oluşturmuştur.

Santoso ve ark. (2016) tarafından yapılan bir çalışmada palmiye ağaçlarının tespiti ve sayımı için basit ve kullanıcı dostu bir yaklaşım sağlanması amaçlanmıştır. Çalışma sonucunda araştırmacılar % 90 ile % 95 arasında doğruluk oranı elde etmişlerdir.

Srestasathiern ve Rakwatin (2014) tarafından önerilen bir yaklaşımda ise ağaçların tespiti lokal tepe noktası tespit hipotezine dayalı olarak yapılmaktadır. Bu yaklaşımda her tepe noktası her bir ağacın en yüksek noktasını göstermekte ve vejetasyon indeksinin ayırt edici özeliğinden faydalanılmaktadır. Bu yaklaşım kullanılarak yaklaşık olarak %90 doğruluk oranına ulaşılmıştır.

Shao ve ark. (2009) tarafından yapılan çalışmalarda toplam biyokütle bağlamında analizler gerçekleştirmiştir. Yazara göre, biyolojik ve ticari nedenlerden dolayı belirli alanlar arasında kalan ağaçların tespit edilmesi büyük önem arz etmektedir. Belirli bir yerde ağaç sayısının tespiti, o bölgedeki söz konusu türlerin üretken kapasitesini yansıtan önemli bir göstergedir. Saha üretken kapasitesi, stantın ağaçtan yetiştirilmesi için gerekli kaynakları tam olarak kullandığı, gelişiminin her aşamasında, belirli bir sahadaki bir stand tarafından üretilen toplam biyokütle (belli bir yetişme alanındaki canlı varlıkların miktarı) anlamına gelmektedir.

Avdan ve ark. (2014) arkeolojik alanda yapmış oldukları çalışmada insansız hava aracı kullanarak elde ettikleri verileri kullanarak çeşitli araştırmalar yapmışlardır. İnsansız hava aracı kullanarak yapmış oldukları çalışmada farklı yükseklik ve farklı bindirme oranları parametrelerini değiştirerek elde edilen görüntüleri işleyerek ortofoto harita üretmişlerdir (Erdoğan, 2016).

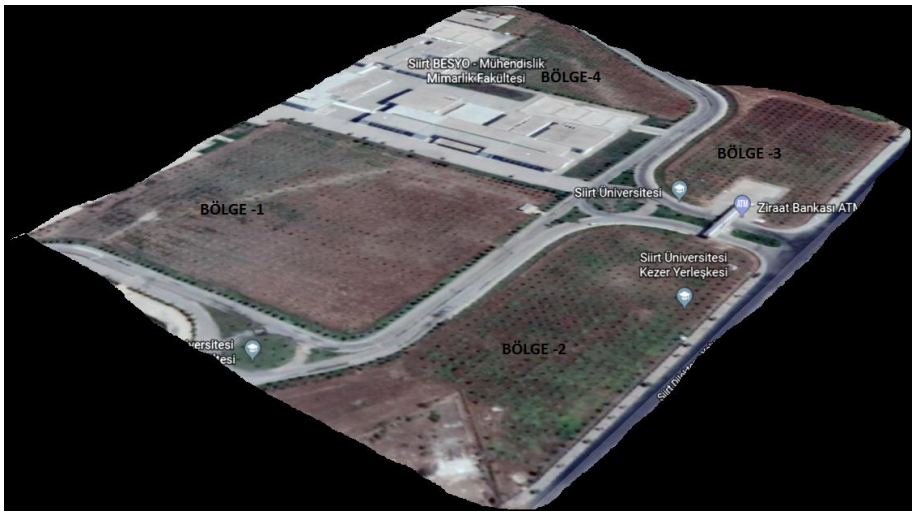
Gürbüz ve ark. (2016) insansız hava araçlarından elde edilen çok yüksek çözünürlüklü renkli (Kırmızı, Yeşil, Mavi) görüntülerden ağaç tespiti yapmıştır. Bunun için öncelikle insansız hava araçları (İHA) ile havadan alınan görüntülerden ortofoto ve otomatik eşleştirme tekniği ile sayısal yüzey modeli (SYM) elde edilmiştir. Oluşturulan ortofotodan ağaç yoğunluğuna göre 4 adet test alanı seçilmiş ve bu test alanlarında ağaçların nesne tabanlı yöntemle segmentasyonu ve sınıflandırması yapılmıştır. Bir sonraki adımda sınıflandırması yapılan görüntülerdeki ağaçların zirve noktaları otomatik olarak tespit edilmiştir. Son olarak ağaçların gerçek konumları (geometrik merkez noktaları) manuel yöntemle elde edilerek bir referans veri oluşturulmuş ve otomatik yöntemle elde edilen ağaçların zirve noktaları (lokal maksimum) ile referans veri karşılaştırılarak doğruluk analizleri yapılmıştır. Elde edilen sonuçlara göre, birinci test alanında % 96, ikinci test alanında % 82, üçüncü test alanında % 96 ve dördüncü test alanında ise % 47 oranında başarı elde edilmiştir.

3. MATERYAL VE METOT

3.1. Görüntü Alımı

Siirt Üniversitesi Kezer Yerleşkesi'nin Google Earth uygulamasından alınmış coğrafi görüntüsü Şekil 3.1' ve 3.2'de gösterilmiştir. Söz konusu alan içerisinde Tablo 3.1'de yaklaşık koordinatları listelenen dört adet bölge çalışma alanı olarak belirlenmiştir. Tablo 3.1'de listelendiği gibi, Bölge-1'in yaklaşık koordinatları; 37 57 47 N, 41 50 57 E, Bölge-2'nin yaklaşık koordinatları; 37 57 38 N, 41 50 55 E, Bölge-3'ün yaklaşık koordinatları; 37 57 44 N, 41 50 08 E ve Bölge-4'ün yaklaşık koordinatları; 37 57 51 N, 41 51 06 E olarak belirlenmiştir.

Şekil 3. 1. Siirt Üniversitesi Kezer Yerleşkesi uydu görüntüsü ve pilot uygulama alanı sınırları



Şekil 3. 2. Siirt Üniversitesi Kezer Yerleşkesi ağaç sayımı için çalışma yapılan bölgeler

Tablo 3. 1. Siirt Üniversitesi Kezer Yerleşkesi Daha Önceden Belirlediğimiz Bölgeler ve Konum Bilgileri

Belirlenmiş Bölgeler	Enlem	Boylam
Bölge-1	41 50 57 E	37 57 47 N
Bölge-2	41 50 55 E	37 57 38 N
Bölge-3	41 50 08 E	37 57 44 N
Bölge-4	41 51 06 E	37 57 51 N

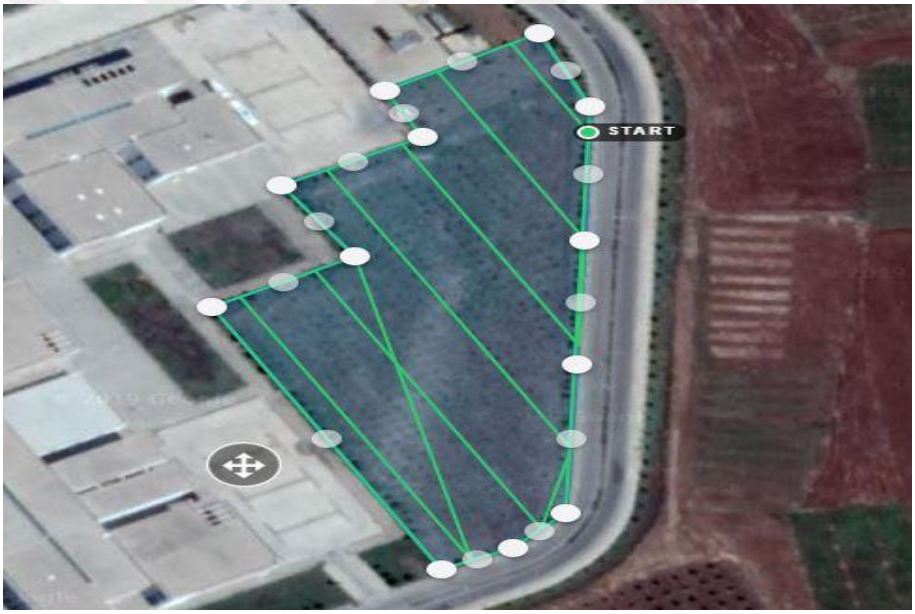
İHA ile çeşitli yüksekliklerden (30 m, 50 m,75 m ve 90 m) 250 adet renkli görüntü çekilmiştir. Bu görüntüler analiz edilerek en iyi resim birleştirme yüksekliği olarak 50 metrelik görüntüler referans olarak kabul edilmiştir.

Arazi çalışmalarında öncelikle ortofoto haritası üretilecek olan alan için Drone Deploy (https://www.dronedeploy.com/) programında görüntülerin çekildiği alanlar belirlenmiştir. Ortofoto yapılacak olan alan üzerinde bulunan tüm alanların belli mesafelerde ve belli yüksekliklerde İHA yardımıyla görüntülerini çekmek için drone deploy yazılımı kullanılmıştır. Tablo 3.1'deki koordinatlar çerçevesinde Drone Deploy programında belirlenen alanlarda uçuşlar gerçekleştirilmiştir (Şekil 3.3, 3.4, 3.5, 3.6).

Şekil 3. 3. Drone Deploy programında Bölge-1 uçuş planlaması

Şekil 3. 4. Drone Deploy programında Bölge-2 uçuş planlaması

Şekil 3. 5. Drone Deploy programında Bölge-3 uçuş planlaması

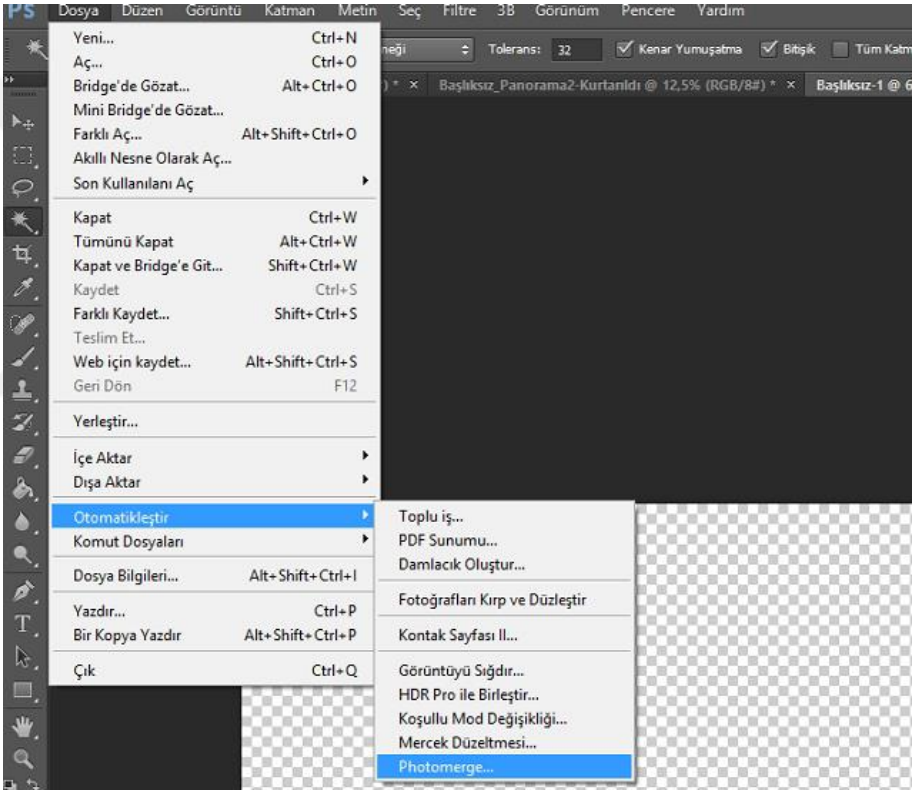


Şekil 3. 6. Drone Deploy programında Bölge-4 uçuş planlaması

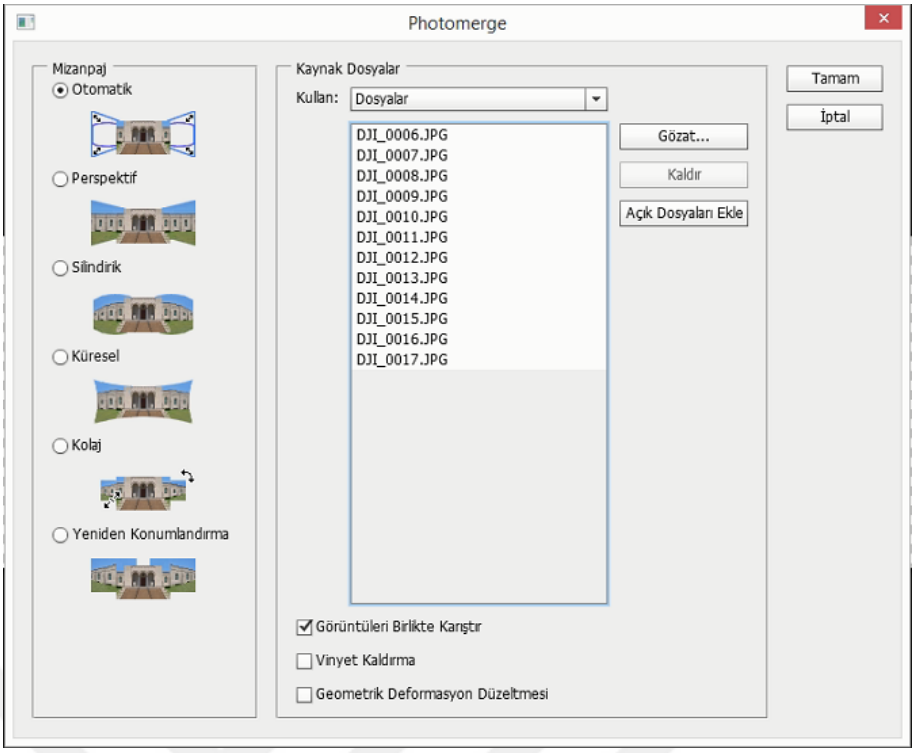
3.2. İHA Görüntülerinden Birleştirilmiş Görüntü Elde Edilmesi (Image Stitching)

Drone Deploy uygulaması varsayılan olarak İHA'nın mekânsal bölgelerde belirlediği yollar üzerindeki uçuşu esnasında belirli zaman aralıklarında ardışık imgeler almasına olanak tanımaktadır. Görüntü birleştirme için ardışık imgelerdeki overlapping (üstüste gelme durumu) oranının %20 üzerinde olması çıkarımını yaptığımız kapsamlı testler sonucunda belirledik. %20 altındaki overlapped imgeler resim birleştirme aşamasında sorunlara neden olmaktadır. Bu nedenle Drone Deploy yazılımı üzerinde

overlapping oranını %30 olarak ayarladık. Yüksek overlapping İHA'nın bölgeyi tarama işleminin daha uzun sürmesine neden olmakla birlikte ardışık görüntü sayısının artması problemi ile karşılaşılır. Ancak resim birleştirme operasyonunun sağlıklı yürüyebilmesi açısından bu durum görmezlikten gelinmiştir. Resim birleştirme (Image Stitching) işlemi Adobe Photoshop (Adobe Systems Inc., San Jose, CA) uygulamasındaki Photomerge menüsü üzerinden gerçekleştirilmiştir. Şekil 3.7.'de Photoshop arayüzünden Photomerge menüsüne erişimin nasıl sağlandığını göstermektedir. Şekil 3.8.'de ise Photmomerge işlemi için resim dosyalarının yüklenmesi gereken ara yüz görülmektedir. Şekil 3.9.'da %30 overlapped imgelerden Photomerge sonucu elde edilmiş birleştirilmiş temsili taranmış bölgeyi göstermektedir.



Şekil 3. 7. Adobe Photoshop Resim Birleştirme Menüsü



Şekil 3. 8. Adobe Photoshop Resim Birleştirme Ara Yüzü İle Resim Dosyalarının Yüklenmesi

Şekil 3. 9. Bölge-3 için Adobe Photoshop Uygulamasının Photomerge Özelliği Kullanılarak Üretilmiş Nihai Görüntüsü

3.3. Görüntü İşleme

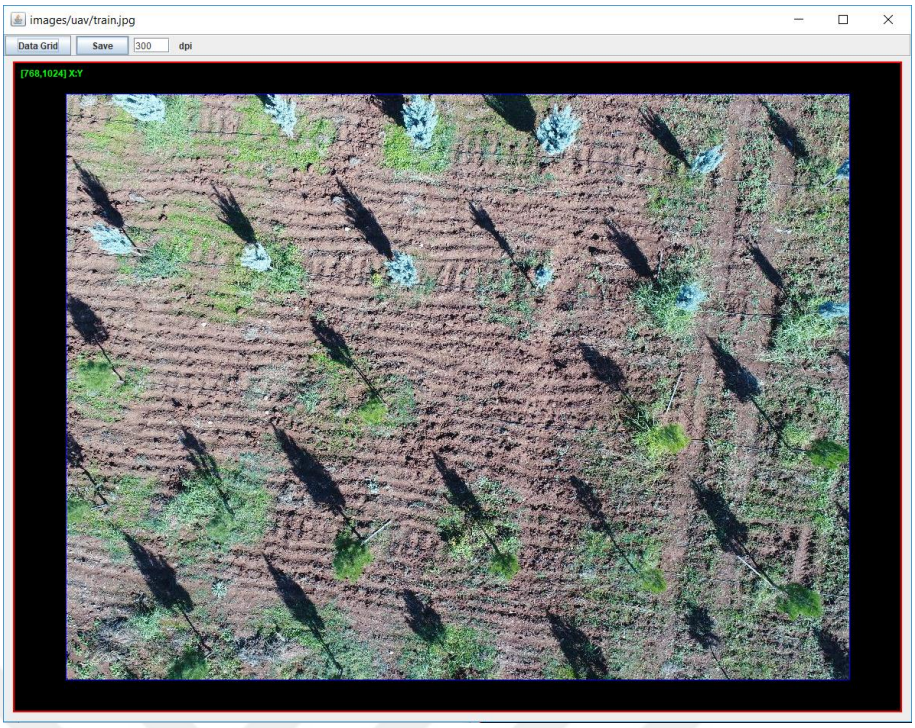
Çalışmamızda ağaç sayısı tespiti için geliştirdiğimiz uygulama temelde imgelerin piksel yoğunluk değerlerine göre seçim yapmaktadır. Bu bağlamda, veri seti üretilirken İHA'nın çektiği ardışık görüntülerden herhangi bir görüntü üzerinden yola çıkılarak veri kümesi hazırlanmıştır. 20 MP'lik görüntüyü işlemek bilgisayar hesaplama maliyeti açısından etkin ve verimli olmadığı için, veri seti için kullanılacak imge 1024x768 çözünürlüğüne indirgenmiştir. Bunun için de Java ekosisteminde çalışan

Açık Cezeri Kütüphanesi (ACK) kullanılmıştır (Ataş, 2016). ACK temelde matris, vektörizasyon, görüntü işleme, makine öğrenmesi ve veri görselleştirmesi için gerekli metotları tek bir nesne üzerinden erişilebilmeye olanak tanıyan El-Cezeri Sibernetik ve Robotik laboratuvarında geliştirilmiş bir yazılım çatısıdır. ACK, metot zincirleme (method chaining) ve akıcı ara yüz (Fluent Interface) tasarım kalıbı temelinde CMatrix nesnesi ile bir static factory method ilkesine göre hareket eder. Programcı CMatrix'in arka planda çağırdığı API'leri bilmesine gerek duymadan CMatrix nesnesi üzerindeki tüm metotları belirli bir akış (pipeline) mantığına göre çağırıp isterse uygulamasını tek satırda bitirebilir. Şekil 3.10.'da imgelerin küçültülmesi için gerekli kod parçasını göstermektedir.

```
/**
 * 1.işlem Çekilen UAV görüntüleri çok büyük olduğundan her birini 1024x768
 * e küçültüyoruz. Sonra bu resimleri Adobe photoshopa auto merge yapması
 * için gönderiyoruz. Veya bu resimleri doğrudan da işleyebiliriz.
 */
public static void resizeImages(String path, int width, int height) {
    for (int i = 0; i < 177; i++) {
        CMatrix cm = CMatrix.getInstance();
        BufferedImage img = ImageProcess.
                readImageFromFile(path + "\\DJI_" + getDigit(i + 6) + ".jpg");
        img = ImageProcess.resizeAspectRatio(img, width, height);
        ImageProcess.saveImage(img, path + "\\resized_images\\" + i + ".jpg");
        System.out.println((i + 6) + ".imge işlendi");
    }
}
```

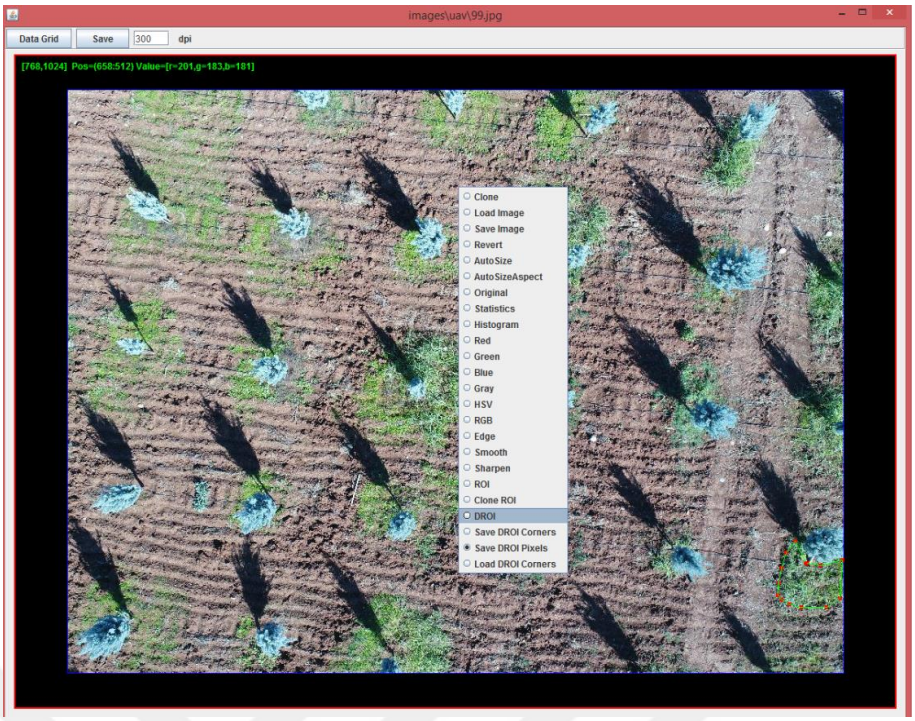
Şekil 3. 10. İmge küçültme işlemi için yazılan metot

Bundan sonra CMatrix.getInstance().imread("images/uav/train.jpg").imshow(); komutu çağrılarak veri seti için ayırdığımız train.jpg imgesini ekranda gösteriyoruz (Şekil 3.11.).

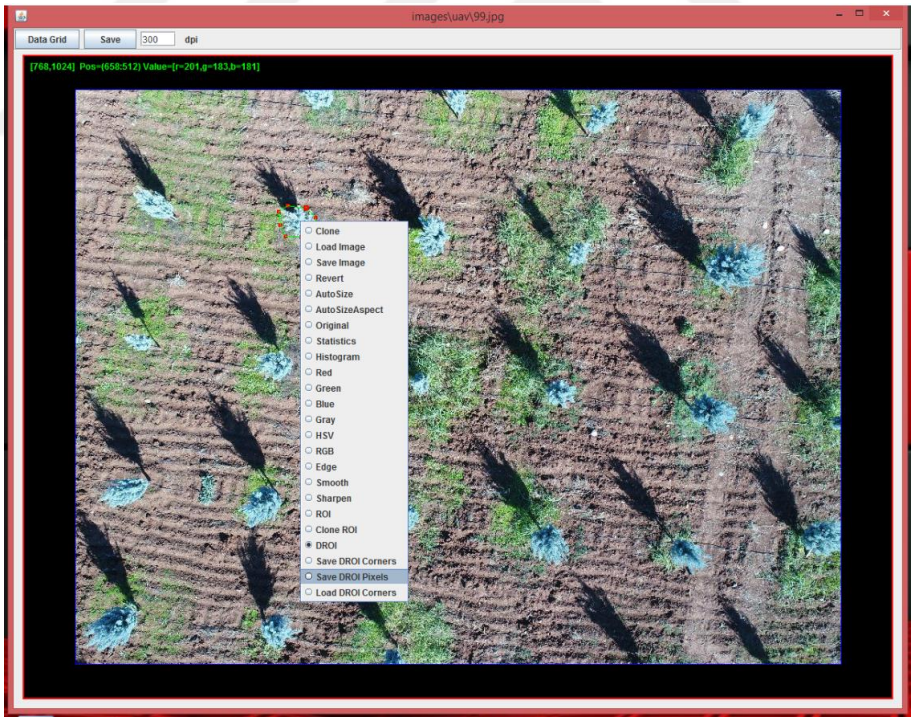


Şekil 3. 11. Küçültülmüş öğrenme veri seti görüntüsü

Belirli bölgelerin işaretlenmesi ve veri kümesi için ilgili sınıfa/gruba ait piksellerin kaydedilebilmesi için Şekil 3.12.'de gösterildiği gibi görüntü üzerinde sağ tıklayıp "DROI" (Dynamic Region of Interest) açılır menüsü aktif edilmelidir. Bu tez çalışmasında, imgelerden 5 farklı sınıf için (ağaç-1, ağaç-2, gölge, çim ve toprak) DROI tanımlamasına gidilmiştir. İstenilen bölge işaretlendikten sonra ilgili bölgenin piksellerinin koordinatlarını kaydetme aşamasına geçilir. Bunun için yine açılır menüden "Save DROI pixels" seçilmelidir (Şekil 3.13.). Bu seçim yapıldıktan sonra ilgili grup/sınıf için bir örnekleme dosyası olarak text veri türünde diske kaydedilir. Daha sonra ilgili grup için kaydedilmiş ***N*** adet örnek dosya birleştirilip, son satırına sınıf etiketi konularak makine öğrenmesinin yapılabilmesine olanak sağlayacak WEKA-ARFF dosya formatına dönüştürülür.



Şekil 3. 12. Açılır menüden DROI ye erişim



Şekil 3. 13. İmge üzerinde ilgi alanının işaretlenmesi sonrası piksellerin kaydedilme işlemi

Şekil 3.14.'te her bir grup/sınıf için dosya birleştirme çağrıları ve çağrılan metodun içeriği gösterilmiştir.

```
//          3.işlem=DROI txt dosyalarını birleştir
        combineFiles("data/uav_2/train","tree_1",10);
        combineFiles("data/uav_2/train","tree_2",10);
        combineFiles("data/uav_2/train","grass",10);
        combineFiles("data/uav_2/train","shadow",10);
        combineFiles("data/uav_2/train","soil",10);
```

```
/**
 * 3.işlem Bir önceki işlemde elde edilen DROI tabanlı pixel konum
 * bilgilerini veri seti üretmek için kullanıyoruz.
 */
public static void combineFiles(String folder, String className, int numOfFiles) {
    String s = "";
    String path = folder + "/" + className;
    for (int i = 1; i <= numOfFiles; i++) {
        String str = FactoryUtils.readFile(path + "/" + className + "_" + i + ".txt");
        s = s + str;
    }
    FactoryUtils.writeToFile(path + "/" + className + ".txt", s);
}
```

Şekil 3. 14. Text dosyalarının birleştirilme işlemi

Bir sonraki işlem, veri setini Weka ARFF formatında üretmektir. Bunun için Şekil 3.15.'te gösterilen program kodu kullanılır.

```
//          4.işlem=DROI piksellerden veri seti üret, açıklama için ilgili metoda gidiniz.
        generateDataSet();
```

```
private static void generateDataSet() {
    CMatrix cm = CMatrix.getInstance().imread("images/uav/train.jpg");
    generateDataSets("data/uav_2/train", "train", "RGB", cm);
    generateDataSets("data/uav_2/train", "train", "HSV", cm);
    generateDataSets("data/uav_2/train", "train", "GRAY", cm);
    generateDataSets("data/uav_2/train", "train", "MIXED", cm);
}
```

```
private static void generateDataSets(String folder, String ds, String modality, CMatrix cm) {
    String str = "";
    str += readPixelsFromFile(folder, "tree_1", "tree_1.txt", modality, cm, 1);
    str += readPixelsFromFile(folder, "tree_2", "tree_2.txt", modality, cm, 2);
    str += readPixelsFromFile(folder, "grass", "grass.txt", modality, cm, 3);
    str += readPixelsFromFile(folder, "soil", "soil.txt", modality, cm, 4);
    str += readPixelsFromFile(folder, "shadow", "shadow.txt", modality, cm, 5);
    String fileName = folder + "/ds" + "_" + modality;
    FactoryUtils.writeToArffFile(fileName + ".arff", str, TLearningType.CLASSIFICATION);
}
```

Şekil 3. 15. Veri seti üretim aşamaları **(Weka ARFF formatı)**

Şekil 3.15.'teki kod bloğu incelendiğinde, her bir grup/sınıf için "*readPixelsFromFile"* metodu üzerinden görüntüdeki piksellerin yoğunluk değerlerinin DROI koordinatları temelinde alınıp makine öğrenmesi algoritmalarının çalışabileceği

WEKA ARFF dosya formatına dönüştürüldüğü görülecektir. "*readPixelsFromFile"* metodu çok uzun olduğu için Ek-1'deki kaynak kodlardan bakılabilir.

3.4. Sınıflandırma

Tez çalışmasında piksel bazlı makine öğrenmesi tekniği kullanılmıştır. Bunun için İHA'dan elde edilen görüntülerden herhangi bir öğrenme seti için ayrılarak, imge üzerinden farklı yerlerden DROI bölgeleri belirlenmiş ve piksel değerleri öznitelik vektörü olarak kullanılmıştır. Bu bağlamda üç farklı modalite (RGB, HSV ve Gray) incelenmiş olup testler yapılmıştır. Öğrenme algoritması olarak, Destek Vektör Makineleri (Support Vector Machine SVM), RepTree, Naive Bayes ve Yapay Sinir Ağ modelinden Çok Katmanlı Perceptron (Multi Layer Perceptron MLP) kullanılmıştır. Söz konusu sınıflandırıcılar Weka uygulamasında varsayılan olarak bulunan sınıflandırıclardır. Açık Cezeri Kütüphanesi (ACK), arka planda Weka kütüphanesini kullanabildiğinden sınıflandırma algoritmaları ACK üzerinden çağrılmıştır. Sınıflandırma performanslarının ölçümleri de aynı şekilde ACK'de mevcut "*FactoryEvaluation"* sınıfı üzerinden sağlanmıştır.

4. BULGULAR VE TARTIŞMA

4.1. Piksel Bazlı 10 Kat Çapraz Doğrulama Sınıflandırma Performansı

Daha önceden de belirtildiği gibi, önerdiğimiz ağaç sayısı kestirim modeli piksel temelli kurgulanmıştır. Öğrenme veri setini üretmek için seçilen bir imge üzerinde her bir sınıfa ait 10 farklı bölgeden Şekil 3.13.'te gösterildiği gibi DROI bölgeleri belirlenmiş ve değişik modaliteler (RGB, HSV, GRAY) için eğitim ve doğrulama (train and validation) veri kümeleri oluşturulmuştur. *RGB, HSV* ve *GRAY* veri setleri toplamda 71581 satırdan meydana gelmektedir. Tablo 4.1.'de kullanılan sınıflandırma algoritmalarının (Naive Bayes, RepTree, SVM, MLP) değişik modalitelerdeki 10 kat çapraz doğrulama performanslarını listelemektedir.

Tablo 4. 1. Değişik sınıflandırıcıların farklı modaliteler deki 10 kat çapraz doğrulama performansları

	% Sınıflandırma başarısı, F-Score			
	Naive Bayes	***RepTree***	***SVM***	***MLP***
GRAY	%48.19 | 0,455	%49.54 | 0,488	%47.67 | 0.433	%45.53 | 0.431
HSV	%85.33 | 0,851	%85.44 | 0.854	%74.56 | 0.741	%87.91 | 0,880
RGB	%59.85 | 0.597	%86.19 | 0.862	%88.47 | 0,885	**%88,89 | 0,889**

Tablo 4.1. incelendiğinde en iyi çapraz doğrulama skoruna RGB formatında MLP sınıflandırıcısı ile ulaşılabildiği görülmektedir. Bunun dışında, GRAY formatının en düşük sınıflandırma performansı sergilediği görülmektedir. Bunun nedeni sınıflar arasındaki farkın gri modalitesinde azaldığı ve önemli oranda bir birine karıştığı (overlapping) gerçeğidir. Diğer taraftan her dört sınıflandırıcı için HSV formatı RGB'ninkine yakın düzeyde sınıflandırma performansı sergilediği anlaşılmaktadır. Ancak RGB, HSV'ye göre daha iyi performans gösterdiğinden arama uzayını karmaşıklığını indirebilmek adına bir sonraki bölümde kullanılmak üzere MLP sınıflandırıcısı ile birlikte temel modalite olarak seçilmiştir.

4.2. Önerilen Ağaç Sayısı Kestirimi Uygulaması

Geliştirilen uygulama gerçek zamanlı (real-time/online) çalışacağı için makine öğrenmesi algoritmasının kendisine sorulan her bir pikselin hangi sınıfa atanacağını

offline sistemlerde olduğu gibi her seferinde öğrenme sürecinden geçmemesi gerekmektedir. Bundan dolayı, *MLP* sınıflandırıcısı bir önceki bölümde çapraz doğrulama için kullandığımız eğitim veri kümesinin tamamı üzerinde eğitilmiştir. Eğitilen *MLP* sınıflandırıcısı Şekil 4.1. ve Şekil 4.2.'deki metot çağrıları üzerinden *JAVA*'daki nesne serileştirmesi (object serialization) yöntemi kullanılarak diske uzantısı "**.cls*" formatında kaydedilmiştir. Söz konusu dosya içeriğinde yapay sinir ağ modelinin topolojisi ve eğitilmiş ağın bağlantı ağırlıklarının tümünü nesnenin orijinal yapısında muhafaza etmektedir. Gerçek zamanlı test yapılacağı zaman uygulama başında ilgili dosya nesne de-serileştirme yapılarak (object deserialization) eğitilmiş *MLP* nesnesine dönüştürülerek zamandan önemli ölçüde tasarruf sağlanır.

```
//        6.işlem=arff dosyasını çeşitli sınıflandırıcıları kullanarak eğit ve kaydet
//        saveModel("data/uav_2/train/ds_rgb.arff", new Bagging(), "data/uav_2/Bagging_ds_rgb.cls");
        saveModel("data/uav_2/train/ds_rgb.arff", new MultilayerPerceptron(), "data/uav_2/MLP_ds_rgb.cls");
```

Şekil 4. 1. Eğitilmiş modelin kaydedilmesini sağlayan metot çağrısı.

```
    private static void saveModel(String pathArff, Classifier model, String pathFile) {
        Instances train = CMatrix.getInstance().readARFF(pathArff).getWekaInstance();
        FactoryEvaluation.saveClassifier(model, train, pathFile);
    }

    public static void saveClassifier(Classifier model, Instances train, String filePath) {
        try {
            Classifier clsCopy = Classifier.makeCopy(model);
            clsCopy.buildClassifier(train);
            weka.core.SerializationHelper.write(filePath, clsCopy);
        } catch (Exception ex) {
            Logger.getLogger(FactoryEvaluation.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
```

Şekil 4. 2. Eğitilmiş sınıflandırıcının diske kaydedilmesini sağlayan kod bloğu

Serileştirilmiş sınıflandırıcı nesnenin diskten okunarak sınıflandırıcı nesneye dönüşebilmesi için Şekil 4.3.'teki metotlar kullanılmaktadır.

```
Classifier cls=FactoryEvaluation.loadClassifier("data/uav_2/MLP_ds_rgb.cls");
```

```
public static Classifier loadClassifier(String filePath) {
    Classifier cls = null;
    try {
        cls = (Classifier) weka.core.SerializationHelper.read(filePath);
    } catch (Exception ex) {
        Logger.getLogger(FactoryEvaluation.class.getName()).log(Level.SEVERE, null, ex);
    }
    return cls;
}
```

Şekil 4. 3. Diskete kayıtlı serileşmiş nesnenin yüklenmesi (Object Deserialization)

Bu işlemden sonra Şekil 4.4.'te gösterilen "*testModel"* metoduna çağrı yapılır. İlgili metot geri dönüş olarak "*CMatrix"* türünde bir nesne geri gönderir. Bu aslında piksellerin sınıflandırıcı tarafından Şekil 4.7.'de gösterildiği gibi uygun gruplara atandığı ve farklı renklerle tanımlandığı bir görüntü üretir.

```
CMatrix cm=testModel("RGB", "images/uav/test.jpg", cls);
cm.imshow();
```

Şekil 4. 4. Test imgesi üzerindeki piksellerin sınıflandırılması için çağrılan metot

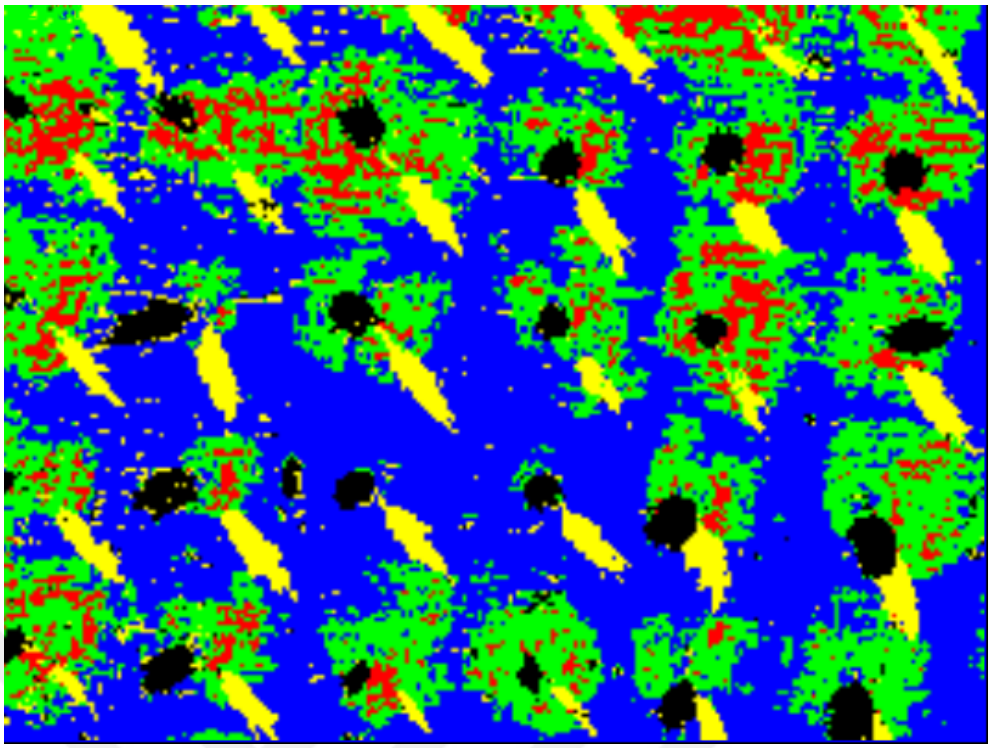
testModel metodu Şekil 4.5.'ten de anlaşılacağı gibi, "*test.jpg*" imgesini *CMatrix* nesnesine yükledikten sonra işlemlerin hızlı yapılabilmesi için görüntüyü 4 te bir oranında küçültür. Daha sonra imge matrisi *cm* üç boyutlu bir diziye *getARGB()* metodu çağrılarak akıtılır. Burada amaç, *RGB* formatındaki imgenin her bir renk kanalına ulaşmaktır. Varsayılan olarak üç boyutlu dizinin ilk boyutu *Alpha, Red, Green ve Blue* piksel değerlerini 8 bit olarak tutar. Bu bilgilerden bir adet test örneği (test instance) yapılır ve "*FactoryEvaluation"* sınıfındaki statik "*performTestWithTrainedClassifier"* metoduna parametre olarak geçilerek bu test örneğinin sınıflandırması istenir.

```
private static CMatrix testModel(String type, String pathImage, Classifier model) {
    CMatrix cm = CMatrix.getInstance().imread(pathImage).imresize(0.25);
    int nr = cm.getRowNumber();
    int nc = cm.getColumnNumber();
    double[][][] predicted_array = new double[4][nr][nc];
    Result res = new Result();
    if (type.equals("RGB")) {
        double[][][] d = cm.imshow().getARGB();
        double[] test_instance = new double[4];
        for (int i = 0; i < nr; i++) {
            for (int j = 0; j < nc; j++) {

                test_instance[0] = d[1][i][j];
                test_instance[1] = d[2][i][j];
                test_instance[2] = d[3][i][j];
                test_instance[3] = 1;
                //Instances test = CMatrix.getInstance(test_instance).getWekaInstance();
                CMatrix.getInstance(test_instance).transpose().toWekaArff("temp.arff", TLearningType.CLASSIFICATION);
                Instances test = CMatrix.getInstance().readARFF("temp.arff").shuffleRowsWeka().getWekaInstance();

                Evaluation eval = FactoryEvaluation.performTestWithTrainedClassifier(model, test, false, false);
                res = res.getResult(eval);
                fillArray(predicted_array, res, i, j);
            }
        }
    }
    return CMatrix.getInstance().fromARGB(predicted_array);
}
```

Şekil 4. 5. testModel metodunun iç yapısı

Bu aşamadan sonra ekranda Şekil 4.6.'daki orijinal test imgesi ve piksellerin sınıflandırıldığı ve sınıflarına göre farklı renklere atandığı Şekil 4.7.'deki görüntü gösterilir.

Şekil 4. 6. Orijinal test resmi (bu resimde yaklaşık 24 adet ağaç sayılabilmektedir)



Şekil 4. 7. Piksel temelli sınıflandırılmış test imgesi

Burada siyah renkler ağaçları, mavi renk toprak sınıfını, sarı renk gölgeleri, yeşil renk çimleri ve yeşil bölgelerde sıklıkla görülen kırmızı lekeler de ağaç-2 sınıfını temsil etmektedir. Bu aşamadan sonra ağaç-1 sınıfının sayısını tespit etmek için Şekil 4.9.'daki gibi renkli görüntü üzerinden sadece siyah pikselleri filtreleme işlemine geçilir. Ve bunun sonucunda elde edilen bölgeler (blobs) tespit edilerek sayım işlemi yapılmaya çalışılır. Bunun için Şekil 4.8.'deki kod parçası kullanılır.

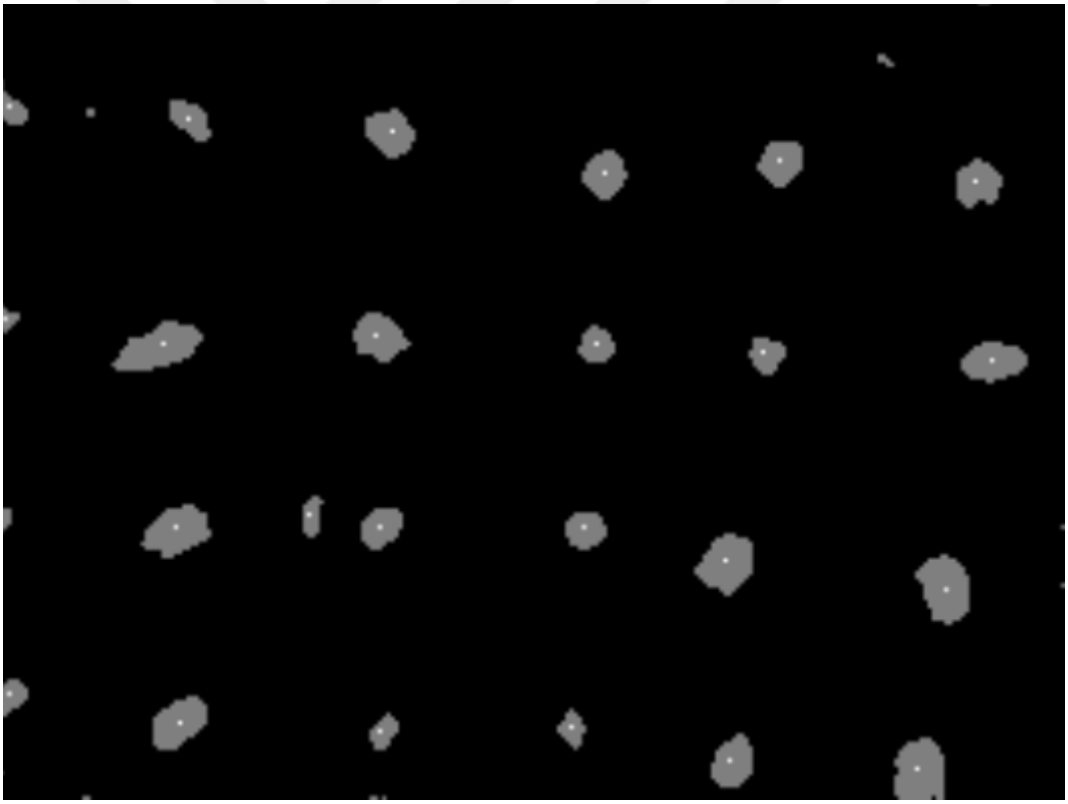
```
double[][][] d = cm.toDoubleArray3D();
int nr = cm.getRowNumber();
int nc = cm.getColumnNumber();
int thr = 20;
for (int i = 0; i < nr; i++) {
    for (int j = 0; j < nc; j++) {
        if (d[1][i][j] > thr || d[2][i][j] > thr || d[3][i][j] > thr) {
            d[1][i][j] = 255;
            d[2][i][j] = 255;
            d[3][i][j] = 255;
        } else {
            d[1][i][j] = 0;
            d[2][i][j] = 0;
            d[3][i][j] = 0;
        }
    }
}
CMatrix cm2 = CMatrix.getInstance()
        .fromARGB(d)
        .rgb2gray()
        .filterMedian(3)
        .imcomplement()
        .binarizeImage()
        .imshow();
```

Şekil 4. 8. ağaç-1 sınıfına ait bölgeleri filtreleyen kod bloğu

Şekil 4. 9. ağaç-1 sınıfına ait bölgeler (blobs)

Şekil 4.9.'daki imge incelendiğinde median filtrenin yok edemediği ancak aynı zamanda ağaç olmayan baz küçük boyutta blobların olduğu görülecektir. Kendi geliştirdiğimiz Blob/Bölge tespit algoritması büyüklüğü belirli bir değerin altındaki

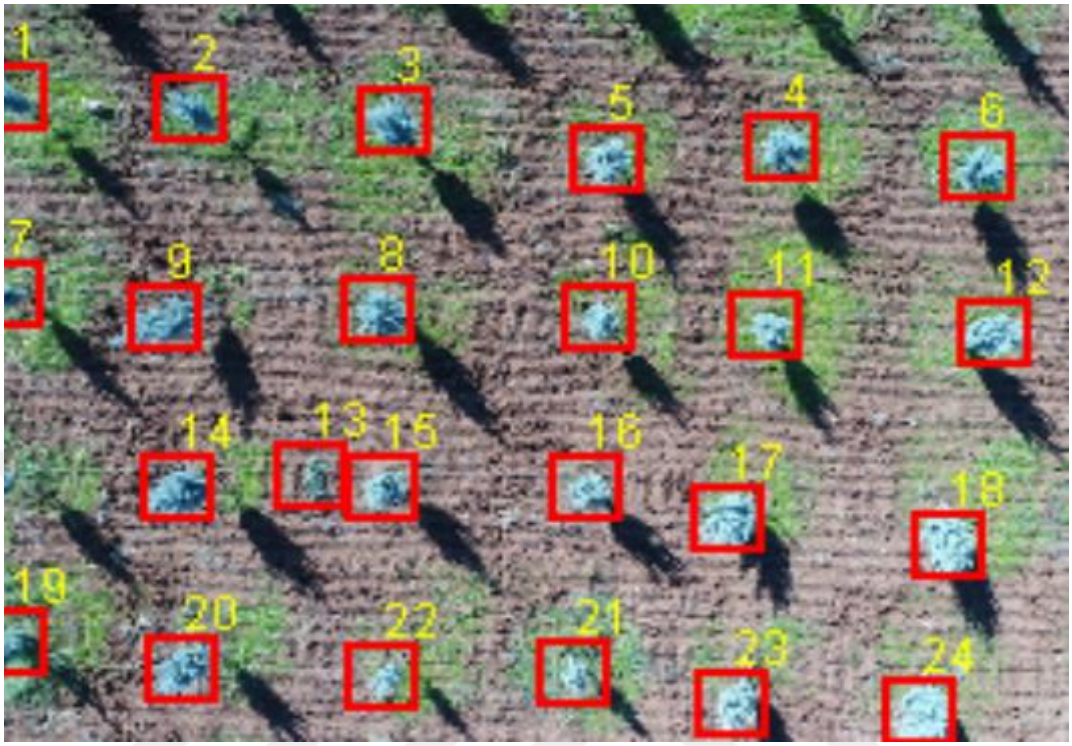
blobları devre dışı bırakarak tespit oranına katkı sağladığı görülmüştür. Blob tespiti algoritması özetle şu şekilde çalışmaktadır. Binary imge sol-üst pozisyondan itibaren bir matris şeklinde (matriste sadece iki değer bulunmaktadır zeminin piksel değeri 0, blobların piksel değerleri ise 255 olarak atanmıştır) aşağıya doğru satır satır taranmaktadır. Algoritma taradığı satırdaki (sağa doğru sütün-sütun giderek) piksellerden 255 değerini bulduğunda bunu yeni bir blob başlangıcı olarak atar. Bir sonraki piksellerde komuşuluk durumuna bakılır şayet 8'li komşuluk (sağ,sol,alt,üst, ve diagoneller) durumu varsa ilgili piksel en uygun bloba atanır. Atandıktan sonra da piksel yoğunluk değeri tekrar kontrol edilmemesi için 255'ten (beyaz), 127 (gri) değerine çekilir. Tüm bu işlemler bittikten sonra blob listesindeki bütün blobların ağırlık merkezleri bulunur. Şekil 4.10.'da tespit edilen blobları ve ağırlık merkezlerini göstermektedir.



Şekil 4. 10. ağaç-1 sınıfına ait bölgeler (blobs)

Daha sonra, ağırlık merkezleri belirlenmiş blobları orijinal resim üzerinde işaretlemeye sora gelir. Bunun için yine *ACK'de* tanımlı *drawRect* ve *drawString* metotları yardımıyla orijinal renkli resim üzerine çizim işlemi (overlay) gerçekleştirilir.

Şekil 4.11.'de orijnal test imgesi üzerine bindirilmiş/çizilmiş ağaç bölgeleri (kırmızı dikdörtgen) ve ağaç sayım numarasını (üstte sarı renkte) göstermektedir.



Şekil 4. 11. Orijinal test görüntüsü üzerine bindirilmiş ağaç yerleri ve numaraları

Tablo 4.2.'de MLP sınıflandırıcısı kullanılarak RGB görüntüsü üzerinden kestirimi yapılan bölgelerdeki doğruluk oranlarını listelemektedir.

Tablo 4. 2. Kezer Yerleşkesinde Daha Önceden Belirlediğimiz Bölgelerdeki Ağaç Sayısı Kestirimi Başarıları

Bölgeler	***Ağaç Sayısı Kestirimi Başarısı***
Bölge-1	%89
Bölge-2	%91
Bölge-3	**%75**
Bölge-4	**%93**
Ortalama	**%87**

Tablo 4.2. incelendiğinde özellikle Bölge-3'te düşük başarı elde edilmesinin sebebi, resim görüntülerinde bulunan cadde ve asfalt gibi nesnelerin sisteme bir sınıf veya grup olarak tanımlanmasının bir sonucu olarak yorumlanabilir.

5. SONUÇ VE ÖNERİLER

5.1. Sonuçlar

Bu tez çalışmasında DJI firmasının Phantom 4 Advanced modeli drone kullanılarak elde edilmiş görüntüler üzerinde Siirt Üniversitesi Kezer yerleşkesinde belirlenmiş 4 farklı bölgedeki ağaç-1 türünün tespiti ve toplam sayısının kestirimi yapılmaya çalışılmıştır. Herhangi bir hazır ticari veya açık erişilebilir uygulama/tool kullanılmadan piksel tabanlı sınıflandırma, filtreleme ve blob tespiti ile ağaç sayılarının resim üzerinde işaretlenmesi JAVA programlama dili ve ACK yazılım çatısı kullanılarak geliştirilmeye çalışılmıştır. Gösterdiği 10 kat çapraz doğrulama performansı ile MLP sınıflandırıcısı ve RGB modalitesi referans olarak seçilmiş ve ağaç sayısı kestiriminde bunlar kullanılmıştır. Ortalama %87 doğruluk oranı ile geliştirilen uygulamanın başarılı olduğu değerlendirilmektedir. Geliştirdiğimiz yazılımın başka nesnelerin örneğin araç, insan, bina sayısının hesaplanmasında ileride yapılacak projelerde kullanılabileceği öngörülmektedir.

5.2. Öneriler

Tez için yapılan çalışma alanında yer alan ağaçların yoğunluk dağılımları, yükseklikleri, gölge boyları, açı değerleri ve konumsal olarak birbirlerine olan yakınlıkları gibi faktörlerin de elde edilen sonuçları doğrudan etkilediği görülmüştür. Çalışma Nisan-Mayıs ayları arasında yapıldığından dolayı bu aylardaki ağaçların ve diğer nesnelerin belirginlik seviyeleri tez sonuçlarına doğrudan etki etmiştir. Çalışmanın yaz aylarında yapılması durumunda o tarihlerdeki toprak ve çim gibi nesnelerin renk özellikleri farklı olacağından dolayı başarı yüzdesinin daha da arttıracağı düşünülmektedir. Çalışma yaptığımız bölgelerde İHA ile çekilen farklı açı ve farklı yüksekliklerde çekilen görüntülerinde ağaçların doğruluk oranına etki ettiği görülmüştür. Örneğin, doğruluk oranının bu durumda olan Bölge-3 olan test alanında % 75 olduğu görülmektedir. Buradaki doğruluk oranın az olmasının sebebi asfalt olan bir nesnenin makine algoritması tarafından bilinmemesidir. Ağaç yoğunluğu az olan ya da homojen alanlarda ise elde edilen sonuçların daha iyi olduğu görülmüştür.

Geliştirilen uygulama yazılımı ileride bu konularda araştırma yapmak isteyen araştırmacılar ve öğrencilerin rahatça kullanımını sağlamak maksadıyla açık kaynak

olarak github üzerinde yayınlanması düşünülmektedir. Ayrıca veri seti üretim aşamaları ve sınıflandırma detayları https://www.youtube.com/watch?reload=9&v=gisBwvAiW-o linkinde kullanıcıların istifadesine açılmıştır. Geliştirdiğimiz yazılımın ağaç nesnesinin dışında örneğin bina, araba, özel nesne vb. sayımını yapabilmesi için ayrıca kullanıcının kod yazmadan etkileşeme geçebileceği bir kullanıcı ara yüzü tasarımı üzerinde çalışmalar yürütmekteyiz. Geliştirilen uygulamanın etkin ve verimli çalışabilmesi için minimum sistem konfigürasyonunuzun intel core i5 işlemci, 8 GB bellek, Windows 8 işletim sistemi, JDK 8 ve geliştirme editörü olarak ta Netbeans 8.1 olması önerilmektedir. Açık Cezeri Kütüphanesinin son sürümlerini indirmek için de https://github.com/hakmesyo/open-cezeri-library kullanabilirsiniz.

6. KAYNAKLAR

Ataş, M., 2016. Open Cezeri Library: A novel java based matrix and computer vision framework. *Computer Applications in Engineering Education*, 24.5: 736-743. https://onlinelibrary.wiley.com/doi/full/10.1002/cae.21745 [Ziyaret Tarihi: 17 Mayıs 2017].

Avdan, U., Şenkal, E., Çömert R., Tuncer S., 2014. İnsansız hava aracı ile oluşturulan verilerin doğruluk analizi, *Uzaktan Algılama ve Coğrafi Bilgi Sistemleri Sempozyumu*, İstanbul, Türkiye 2014.

Bazi, Y., Malek, S., Alajlan, N., AlHichri, H., 2014. An automatic approach for palm tree counting in UAV images, *IEEE Geoscience and Remote Sensing Symposium* pp. 537-540.

Colomina, I., Molina P., 2014. Unmanned Aerial System for Photogrammetry and Remote Sensing, *ISPRS Journal Of Photogrammetry And Remote Sensing.*

Colomina, I., Molina, P., 2014. Unmanned aerial systems for photogrammetry and remote sensing: A review, *ISPRS Journal of photogrammetry and remote sensing*, *92*, 79-97.

Cryderman, C., Mah, S. B., Shufletoski, A., 2014. Evaluation of UAV photogrammetric accuracy for mapping and earthworks computations, *Geomatica*, *68*(4), 309-317.

Draeyer, B. and Strecha, C., 2014. White paper: How accurate are UAV surveying methods, *Pix4D White paper*.

Gürbüz, M.F., 2016. "Kentsel Alanlarda İha Görüntülerinden Ortofoto Oluşturma Ve Otomatik Ağaç Tespiti", Yüksek Lisans Tezi, Hacettepe Üniversitesi, *Fen Bilimleri Enstitüsü*.

Mohan, M., Silva, C., Klauberg, C., Jat, P., Catts, G., Cardil, A., Hudak, A.T., Dia, M., 2017. Individual tree detection from unmanned aerial vehicle (UAV) derived canopy height model in an open canopy mixed conifer forest. *Forests*, 8(9), 340.

Moranduzzo, T. and Melgani, F., 2013. Automatic car counting method for unmanned aerial vehicle images. *IEEE Transactions on Geoscience and remote sensing*, 52(3), 1635-1647.

Santoso, H., Tani, H., Wang, X., 2016. A simple method for detection and counting of oil palm trees using high-resolution multispectral satellite imagery, *International journal of remote sensing, 37*(21), 5122-5134.

Shafri, H. Z., Hamdan, N., Saripan, M. I., 2011. Semi-automatic detection and counting of oil palm trees from high spatial resolution airborne imagery, *International journal of remote sensing, 32*(8), 2095-2115.

Shao, Y., Raupach, M. R., Leys, J. F., 1996. A model for predicting aeolian sand drift and dust entrainment on scales from paddock to region, *Soil Research, 34*(3), 309-342.

Srestasathiern, P., Rakwatin, P., 2014. Oil palm tree detection with high resolution multi-spectral satellite imagery, *Remote Sensing*, 6(10), 9749-9774.

Tao, C. V., Hu, Y., Jiang, W., 2004. Photogrammetric exploitation of IKONOS imagery for mapping applications, *International journal of remote sensing, 25*(14), 2833-2853.

Torun, A., 2016. Integrating Geospatial Technologies: Reflections on Intergeo 2016, *GIM International*.

Torun, A., 2017. İnsansız Hava Aracı (İHA) Sektöründe Trend: İHA Fotogrametrisi Bakışıyla, *Afyon Kocatepe Üniversitesi Fen Ve Mühendislik Bilimleri Dergisi, 17*(4), 35-52.

Yılmaz, V., Akar, A., Akar, Ö., Güngör, O., Karslı, F., Gökalp, E., 2013. İnsansiz hava araci ile üretilen ortofoto haritalarda doğruluk analizi, *Türkiye Ulusal Fotogrametri ve Uzaktan Algılama Birliği VII. Teknik Sempozyumu (TUFUAB'2013)*.

Url-1 < https://www.adobe.com/tr/ >, [Ziyaret Tarihi: 1 Temmuz 2019].

Url-2 < https://www.dronedeploy.com/ >, [Ziyaret Tarihi: 1 Temmuz 2019].

EKLER

EK-1 Geliştirilen Uygulamanın Kaynak Kodları.

```
//*******************************************
//MainProgram.java
//*******************************************
	package yeni_kodlar;

	import cezeri.evaluater.FactoryEvaluation;
	import cezeri.image_processing.ImageProcess;
	import cezeri.matrix.CMatrix;
	import cezeri.types.TLearningType;
	import cezeri.utils.FactoryUtils;
	import java.awt.Point;
	import java.awt.image.BufferedImage;
	import java.io.BufferedReader;
	import java.io.File;
	import java.io.FileReader;
	import java.io.IOException;
	import java.util.ArrayList;
	import java.util.List;
	import eski_kodlar.utils.Utils;
	import static eski_kodlar.utils.Utils.getDigit;
	import java.awt.Color;
	import java.util.Arrays;
	import java.util.Collections;
	import weka.classifiers.Classifier;
	import weka.classifiers.Evaluation;
	import weka.classifiers.bayes.NaiveBayes;
	import weka.classifiers.functions.MultilayerPerceptron;
	import weka.classifiers.functions.SMO;
	import weka.classifiers.meta.Bagging;
	import weka.classifiers.trees.REPTree;
	import weka.core.Instances;

	public class MainProgram {

	   public static void main(String[] args) {
	//	1.işlem=imgeleri küçült, açıklama için ilgili metoda gidiniz.
	//	resizeImages("C:\\Users\\elcezerilab\\Desktop\\100MEDIA", 1024, 768);
	//
	//	  2.işlem=DROI almak için imgeyi görüntüle, açıklama için ilgili metoda
gidiniz.
	//	showImages();
	//
	//	3.işlem=DROI txt dosyalarını birleştir
	//	combineFiles("data/uav_2/train","tree_1",10);
	//	combineFiles("data/uav_2/train","tree_2",10);
```

```
        //      combineFiles("data/uav_2/train","grass",10);
        //      combineFiles("data/uav_2/train","shadow",10);
        //      combineFiles("data/uav_2/train","soil",10);
        //
        //          4.işlem=DROI piksellerden veri seti üret, açıklama için ilgili metoda
gidiniz.
        //      generateDataSet();
        //
        //       5.işlem=Arff dosyasını offline cross valide et, açıklama için ilgili metoda
gidiniz.
        //      evaluateModel("data/uav_2/train/ds_mixed.arff", new SMO());
        //                          evaluateModel("data/uav_2/train/ds_mixed.arff",    new
MultilayerPerceptron());
        //      evaluateModel("data/uav_2/train/ds_mixed.arff", new REPTree());
        //      evaluateModel("data/uav_2/train/ds_mixed.arff", new SMO());
        //      evaluateModel("data/uav_2/train/ds_mixed.arff", new NaiveBayes());
        //      evaluateModel("data/uav_2/train/ds_mixed.arff", new SMO());

        //      6.işlem=arff dosyasını çeşitli sınıflandırıcıları kullanarak eğit ve kaydet
        //                  saveModel("data/uav_2/train/ds_rgb.arff", new Bagging(),
"data/uav_2/Bagging_ds_rgb.cls");
            saveModel("data/uav_2/train/ds_rgb.arff",    new    MultilayerPerceptron(),
"data/uav_2/MLP_ds_rgb.cls");
        //
        //          7.işlem=Önceden yapılmış arff dosyasını train olarak kullanıp, resim
dosyasının pixellerini sınıflandır.
            Classifier
cls=FactoryEvaluation.loadClassifier("data/uav_2/MLP_ds_rgb.cls");
            CMatrix cm=testModel("RGB", "images/uav/test.jpg", cls);
            cm.imshow();

        //      8.işlem ara resimde siyah bölgelerin sayısını hesapla
            countTreeNumbers(cm);

        }

        /**
         * 1.işlem Çekilen UAV görüntüleri çok büyük olduğundan her birini
1024x768
         * e küçültüyoruz. Sonra bu resimleri Adobe photoshopa auto merge yapması
         * için gönderiyoruz. Veya bu resimleri doğrudan da işleyebiliriz.
         */
        public static void resizeImages(String path, int width, int height) {
            for (int i = 0; i < 177; i++) {
                CMatrix cm = CMatrix.getInstance();
                BufferedImage img = ImageProcess.
                        readImageFromFile(path + "\\DJI_" + getDigit(i + 6) + ".jpg");
                img = ImageProcess.resizeAspectRatio(img, width, height);
                ImageProcess.saveImage(img, path + "\\resized_images\\" + i + ".jpg");
                System.out.println((i + 6) + ".imge işlendi");
```

```
        }
    }

    /**
     * 3.işlem Bir önceki işlemde elde edilen DROI tabanlı pixel konum
     * bilgilerini veri seti üretmek için kullanıyoruz.
     */
    public static void combineFiles(String folder, String className, int numOfFiles) {
        String s = "";
        String path = folder + "/" + className;
        for (int i = 1; i <= numOfFiles; i++) {
            String str = FactoryUtils.readFile(path + "/" + className + "_" + i + ".txt");
            s = s + str;
        }
        FactoryUtils.writeToFile(path + "/" + className + ".txt", s);
    }

    /**
     * 2.işlem herhangi bir resmi train olarak başka bir resmi de test olarak
     * belirliyoruz. Sonra bu resimleri CMatrix
     *  cm=CMatrix.getInstance().imread(path).imshow(); komutu ile açtıktan sonra
     * DROI ile öğrenmesini istediğimiz her bir sınıftan en az 10 veya 20 farklı
     * bölgeden Dinamik Reigion of Interest yani DROI belirleyip kaydediyoruz.
     * DROI belirlemek için imshow ile gösterilen resmin üzerine sağ tıklayıp
     * DROI kısmına gelerek mouse ile ilgi bölgelerini işaretliyoruz. sonra da
     * saveDROI pixelsi seçerek ve uygun bir isimle mesela tree_1.txt olarak
     * kaydediyoruz. Bir sonraki adımda bu txtleri merge ederek tek bir veriseti
     * üreteceğiz.
     */
    public static void showImages() {
        CMatrix cm_train = CMatrix.getInstance().imread("images/uav/train.jpg").imshow();
        CMatrix cm_test = CMatrix.getInstance().imread("images/uav/test.jpg").imshow();
    }

    private static void generateDataSet() {
        CMatrix cm = CMatrix.getInstance().imread("images/uav/train.jpg");
        generateDataSets("data/uav_2/train", "train", "RGB", cm);
        generateDataSets("data/uav_2/train", "train", "HSV", cm);
        generateDataSets("data/uav_2/train", "train", "GRAY", cm);
        generateDataSets("data/uav_2/train", "train", "MIXED", cm);
    }

    private static void generateDataSets(String folder, String ds, String modality, CMatrix cm) {
        String str = "";
```

```
            str += readPixelsFromFile(folder, "tree_1", "tree_1.txt", modality, cm, 1);
            str += readPixelsFromFile(folder, "tree_2", "tree_2.txt", modality, cm, 2);
            str += readPixelsFromFile(folder, "grass", "grass.txt", modality, cm, 3);
            str += readPixelsFromFile(folder, "soil", "soil.txt", modality, cm, 4);
            str += readPixelsFromFile(folder, "shadow", "shadow.txt", modality, cm,
5);
            String fileName = folder + "/ds" + "_" + modality;
            FactoryUtils.writeToArffFile(fileName + ".arff", str,
TLearningType.CLASSIFICATION);
        }

        public static String readPixelsFromFile(String folder, String className,
String txtFileName, String modality, CMatrix cm, int classID) {
            String filePath = folder + "/" + className + "/" + txtFileName;
            File file = new File(filePath);
            if (!file.exists()) {
                return "";
            }
            int r = 0;
            int c = 0;
            String s = "";
            String ret = "";

            if (modality.equals("GRAY")) {
                double[][] d = cm.rgb2gray().toDoubleArray2D();
                try (BufferedReader br = new BufferedReader(new
FileReader(filePath))) {
                    while ((s = br.readLine()) != null) {
                        String[] row = s.split(",");
                        r = Integer.parseInt(row[0]);
                        c = Integer.parseInt(row[1]);
                        ret += d[r][c] + "," + classID + "\n";
                    }
                } catch (IOException e) {
                    e.printStackTrace();
                    return "";
                }
            } else if (modality.equals("RGB")) {
                double[][][] d = cm.getARGB();
                try (BufferedReader br = new BufferedReader(new
FileReader(filePath))) {
                    while ((s = br.readLine()) != null) {
                        String[] row = s.split(",");
                        r = Integer.parseInt(row[0]);
                        c = Integer.parseInt(row[1]);
                        ret += d[1][r][c] + "," + d[2][r][c] + "," + d[3][r][c] + "," + classID +
"\n";
                    }
                } catch (IOException e) {
                    e.printStackTrace();
```

```
                return "";
            }
        } else if (modality.equals("HSV")) {
            double[][][] d = cm.rgb2hsv().getARGB();
            try (BufferedReader br = new BufferedReader(new
FileReader(filePath))) {
                while ((s = br.readLine()) != null) {
                    String[] row = s.split(",");
                    r = Integer.parseInt(row[0]);
                    c = Integer.parseInt(row[1]);
                    ret += d[1][r][c] + "," + d[2][r][c] + "," + d[3][r][c] + "," + classID +
"\n";
                }
            } catch (IOException e) {
                e.printStackTrace();
                return "";
            }
        } else if (modality.equals("MIXED")) {
            double[][] d_gray = cm.rgb2gray().toDoubleArray2D();
            double[][][] d_rgb = cm.getARGB();
            double[][][] d_hsv = cm.rgb2hsv().getARGB();
            try (BufferedReader br = new BufferedReader(new
FileReader(filePath))) {
                while ((s = br.readLine()) != null) {
                    String[] row = s.split(",");
                    r = Integer.parseInt(row[0]);
                    c = Integer.parseInt(row[1]);
                    ret += d_gray[r][c] + "," + d_rgb[1][r][c] + "," + d_rgb[2][r][c] + ","
+ d_rgb[3][r][c] + "," + d_hsv[1][r][c] + "," + d_hsv[2][r][c] + "," + d_hsv[3][r][c] + ","
+ classID + "\n";
                }
            } catch (IOException e) {
                e.printStackTrace();
                return "";
            }
        }

        return ret;
    }

    public static void evaluateModel(String pathTrain, Classifier model) {
        Instances train =
CMatrix.getInstance().readARFF(pathTrain).getWekaInstance();
        FactoryEvaluation.performCrossValidate(model, train, 10, true, false);
    }

    private static CMatrix testModel(String type, String pathImage, Classifier
model) {
        CMatrix cm = CMatrix.getInstance().imread(pathImage).imresize(0.25);
        int nr = cm.getRowNumber();
```

```
            int nc = cm.getColumnNumber();
            double[][][] predicted_array = new double[4][nr][nc];
            Result res = new Result();

            if (type.equals("GRAY")) {
        //        double[][] d = cm.rgb2gray().toDoubleArray2D();
        //        double[] test_instance = new double[2];
        //        for (int i = 0; i < nr; i++) {
        //           for (int j = 0; j < nc; j++) {
        //              test_instance[0] = d[i][j];
        //              test_instance[1] = -1;
        //                                                          Instances test =
CMatrix.getInstance(test_instance).getWekaInstance();
        //                                                       Evaluation eval =
FactoryEvaluation.performTestWithTrainedClassifier(model, test, true, false);
        //              res = res.getResult(eval);
        //              //fillArray(predicted_array, res, i, j);
        //           }
        //        }

            } else
            if (type.equals("RGB")) {
               double[][][] d = cm.imshow().getARGB();
               double[] test_instance = new double[4];

               for (int i = 0; i < nr; i++) {
                  for (int j = 0; j < nc; j++) {

                     test_instance[0] = d[1][i][j];
                     test_instance[1] = d[2][i][j];
                     test_instance[2] = d[3][i][j];
                     test_instance[3] = 1;
                     //Instances                          test                          =
CMatrix.getInstance(test_instance).getWekaInstance();

CMatrix.getInstance(test_instance).transpose().toWekaArff("temp.arff",
TLearningType.CLASSIFICATION);
                     Instances                          test                          =
CMatrix.getInstance().readARFF("temp.arff").shuffleRowsWeka().getWekaInstance();

                     Evaluation                         eval                          =
FactoryEvaluation.performTestWithTrainedClassifier(model, test, false, false);
                     res = res.getResult(eval);
                     fillArray(predicted_array, res, i, j);

                  }
               }
            } else if (type.equals("HSV")) {

            } else if (type.equals("MIXED")) {
```

```
            }
            return CMatrix.getInstance().fromARGB(predicted_array);
        }

        private static void saveModel(String pathArff, Classifier model, String pathFile) {
            Instances train = CMatrix.getInstance().readARFF(pathArff).getWekaInstance();
            FactoryEvaluation.saveClassifier(model, train, pathFile);
        }

        private static void fillArray(double[][][] d, Result res, int i, int j) {
            if (res.predictedClassID == 1) {
                d[0][i][j] = 255;
                d[1][i][j] = 255;
                d[2][i][j] = 0;
                d[3][i][j] = 0;
            } else if (res.predictedClassID == 2) {
                d[0][i][j] = 255;
                d[1][i][j] = 0;
                d[2][i][j] = 255;
                d[3][i][j] = 0;
            } else if (res.predictedClassID == 3) {
                d[0][i][j] = 255;
                d[1][i][j] = 0;
                d[2][i][j] = 0;
                d[3][i][j] = 255;
            } else if (res.predictedClassID == 4) {
                d[0][i][j] = 255;
                d[1][i][j] = 255;
                d[2][i][j] = 255;
                d[3][i][j] = 0;
            } else if (res.predictedClassID == 5) {
                d[0][i][j] = 255;
                d[1][i][j] = 0;
                d[2][i][j] = 255;
                d[3][i][j] = 255;
            }

        }

        private static CMatrix countTreeNumbers(CMatrix cm) {
        //    CMatrix cm = CMatrix.getInstance().imread("images/uav/ara.jpg").imshow();
            double[][][] d = cm.toDoubleArray3D();
            int nr = cm.getRowNumber();
            int nc = cm.getColumnNumber();
            int thr = 20;
            for (int i = 0; i < nr; i++) {
```

```
    for (int j = 0; j < nc; j++) {
        if (d[1][i][j] > thr || d[2][i][j] > thr || d[3][i][j] > thr) {
            d[1][i][j] = 255;
            d[2][i][j] = 255;
            d[3][i][j] = 255;
        } else {
            d[1][i][j] = 0;
            d[2][i][j] = 0;
            d[3][i][j] = 0;
        }
    }
}
CMatrix cm2 = CMatrix.getInstance()
        .fromARGB(d)
        .rgb2gray()
        .filterMedian(3)
        .imcomplement()
        .binarizeImage()
        .imshow();

double[][] d2 = cm2.toDoubleArray2D();
List<List<Point>> lst = new ArrayList();
List<List<Point>> lst2 = new ArrayList();
List<List<Point>> lst3 = new ArrayList();
for (int i = 0; i < nr; i++) {
    for (int j = 0; j < nc; j++) {
        if (d2[i][j] == 255) {
            if (!isNeighbor(i, j, lst)) {
                List<Point> lstBlob = new ArrayList();
                lstBlob.add(new Point(j, i));
                lst.add(lstBlob);
                lst2.add(lstBlob);
            }
        }
    }
}
for (List<Point> list : lst) {
    for (Point point : list) {
        d2[point.y][point.x] = 127;
    }
}
cm2 = cm2.setArray(d2);
//

lst=removeTinyBlobs(lst);
//lst = sortList(lst);

List<Point> temp = new ArrayList();
for (int i = 0; i < lst.size(); i++) {
    List<Point> list = lst.get(i);
```

```
    List<Point> next = null;
    if (i < lst.size() - 1) {
       next = lst.get(i + 1);
    }
    if (next != null) {
       List<Point> merged = new ArrayList();
       if (canBlobMerge(list, next, merged)) {
          i++;
          list = copy(merged);
          if (i < lst.size() - 1) {
             next = lst.get(i + 1);
          }else{
             next=null;
          }
          if (next!=null && canBlobMerge(list, next, merged)) {
             i++;
             list = copy(merged);
             if (i < lst.size() - 1) {
                next = lst.get(i + 1);
             }else{
                next=null;
             }
             if (next!=null && canBlobMerge(list, next, merged)) {
                i++;
                list = copy(merged);
                if (i < lst.size() - 1) {
                   next = lst.get(i + 1);
                }else{
                   next=null;
                }
                if (next!=null && canBlobMerge(list, next, merged)) {
                   i++;
                }
             }
          }
          lst3.add(merged);
          //temp = next;

       } else {
          if (temp != list && list.size() > 10) {
             lst3.add(list);
          }
       }

    } else {
       if (temp != list && list.size() > 10) {
          lst3.add(list);
       }
    }
}
```

```
            List<Point> merkezList = centerBlob(lst3);

            double[][] d3 = cm2.toDoubleArray2D();
            for (Point p : merkezList) {
               d3[p.y][p.x] = 255;
            }
            System.out.println("tree count = " + merkezList.size());
            cm2 = cm2.setArray(d3)
                  .imshow();

            CMatrix cm3 = CMatrix.getInstance().imread("images/uav/test.jpg").imresize(0.25);
            int nk=0;
            for (Point p : merkezList) {
              cm3=cm3.drawRect(p.y-8, p.x-8, 16, 16, 2, Color.red).drawString(""+(++nk), p.y-10, p.x, 0, Color.yellow);
            }
            cm3.imshow();

            return cm3;
         }

         private static boolean isNeighbor(int i, int j, List lst) {
            for (Object obj : lst) {
               List<Point> lstBlob = (List<Point>) obj;
               for (Point point : lstBlob) {
                  if ((Math.abs(i - point.y) <= 1 && Math.abs(j - point.x) <= 1) || Math.abs(i - (point.y - 1)) <= 0 || (Math.abs(i - (point.y - 1)) <= 0 && Math.abs(j - (point.x - 1)) <= 0)) {
                     lstBlob.add(new Point(j, i));
                     return true;
                  }
               }
            }
            return false;
         }

         private static boolean canBlobMerge(List<Point> current, List<Point> next, List<Point> merged) {
            boolean ret = false;
            for (Point p1 : current) {
               for (Point p2 : next) {
                  if (near(p1, p2)) {
                     for (Point p : current) {
                        merged.add(p);
                     }
                     for (Point p : next) {
                        merged.add(p);
                     }
```

```
                return true;
            }
        }
    }
    return ret;
}

private static boolean near(Point p1, Point p2) {
    boolean ret = false;
    if (Math.abs(p1.x - p2.x) <= 1 && Math.abs(p1.y - p2.y) <= 1) {
        ret = true;
    }
    return ret;
}

private static List<Point> copy(List<Point> merged) {
    List<Point> ret = new ArrayList();
    for (Point p : merged) {
        ret.add(new Point(p.x, p.y));
    }
    return ret;
}

private static List<List<Point>> sortList(List<List<Point>> lst) {
    List<List<Point>> ret = new ArrayList();
    List<Point> ls = centerBlob(lst);
    List<Double> dstx = distanceList(ls);
    List<Double> dst = distanceList(ls);
    Collections.sort(dst);
    for (Double d : dst) {
        for (int i = 0; i < dstx.size(); i++) {
            if (Math.round(d) == Math.round(dstx.get(i))) {
                ret.add(copy(lst.get(i)));
            }
        }
    }
    return ret;
}

private static List<Point> centerBlob(List<List<Point>> lst) {
    List<Point> merkezList = new ArrayList();
    for (List<Point> ls : lst) {
        int tx = 0;
        int ty = 0;
        for (Point p : ls) {
            tx += p.x;
            ty += p.y;
        }
        tx = tx / ls.size();
        ty = ty / ls.size();
```

```
            merkezList.add(new Point(tx, ty));
        }
        return merkezList;
    }

    private static List<Double> distanceList(List<Point> ls) {
        List<Double> ret = new ArrayList();
        for (Point p : ls) {
            ret.add(Math.sqrt(p.x * p.x + p.y * p.y));
        }
        return ret;
    }

    private static List<List<Point>> removeTinyBlobs(List<List<Point>> lst) {
        List<List<Point>> ret=new ArrayList();
        for (List<Point> ls : lst) {
            if (ls.size()>15) {
                ret.add(copy(ls));
            }
        }
        return ret;
    }

}
```

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Ayhan TALAY
Doğum Yeri ve Tarihi : ÇATAK 12.11.1987
Telefon : 0544-961-05-27
E-posta : atalay6565@gmail.com

EĞİTİM

Derece			**Bitirme Yılı**
Lise	:	Şehit İbrahim Karaoğlanoğlu Lisesi	2003
Üniversite	:	Azerbaycan Teknik Üniversitesi Bilgisayar Mühendisliği	2009

İŞ DENEYİMLERİ

Yıl	**Kurum**	**Görevi**
2012-Devam Ediyor	Siirt Üniversitesi, Bilgi İşlem Daire Başkanlığı	Müh.

UZMANLIK ALANI : Network , Sistem Sunucu Güvenliği

BİLDİRİ : International Conference on Multidisciplinary, Science, Engineering and Technology (IMESET'17 Bitlis) Oct 27-29, 2017, Bitlis